SANDIA REPORT

SAND2005-XXXX
Unlimited Release
Printed April 26, 2005

An Introduction to Numerical Analysis for Computational Fluid Mechanics

S. Scott Collis (Sandia National Laboratories)

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/ordering.htm

AN INTRODUCTION TO NUMERICAL ANALYSIS FOR COMPUTATIONAL FLUID MECHANICS

By

S. Scott Collis
Computational Mathematics & Algorithms
Sandia National Laboratories [1]

April 26, 2005

[1]Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Acknowledgments

These notes were originally assembled for use in MECH 676 Computational Fluid Mechanics, which I taught at Rice University. In writing these notes, I have made extensive use of my class notes from Prof. Parviz Moin's course in numerical analysis at Stanford University. Since that time, Prof. Moin has published his class notes as the *Fundamentals of Engineering Numerical Analysis*, Cambridge University Press, 2001. Readers are strongly encouraged to consult this text. However, I believe that the focus of the current notes on computational fluid mechanics nicely complements Moin's work and future versions of this document will be further specialized to methods for problems in fluid mechanics. As usual, all mistakes in the presentation are mine and comments/suggestions can be sent to `sscoll@sandia.gov`

Table of Contents

Chapter 1

Introduction

Computational Fluid Dynamics, or CFD for short, is a broad reaching field that utilizes advanced methods of numerical analysis to solve problems in continuum mechanics. These problems may include the determination of lift and drag on an aircraft or car; simulation of turbulent flow and heat transfer in a jet engine; the study of the atmospheric boundary layer; or even the interstellar wind. The fundamental similarity in each of these examples is that the physical process can be modeled, at some level of approximation, as a continuum of fluid particles and the equations which represent the continuum dynamics can be approximately solved using a computer. It is important to point out, however, that the appropriate mathematical models and numerical methods may be very different for each problem.

In the course of these notes, we will present the fundamental concepts and methods of scientific computing required to utilize, analyze, and develop CFD algorithms. This course will *not* be an exhaustive review of CFD algorithms nor will we discuss many of the state-of-the-art methods currently in use. The number of unique methods and their complexity make that an impossible task for a single semester course. Instead, we will focus on the fundamentals of scientific computing and apply these fundamentals to study selected CFD algorithms. The emphasis will be on understanding the methods of analysis so that you can interpret, select, and/or design appropriate algorithms to solve the inevitably more and more complex problems that you will encounter in your careers.

A consistent theme throughout this course will be the identification and estimation of errors in approximation. Just like experimental measurements, there are sources of uncertainty and error in numerical simulations. These include: mathematical approximations of the physical process, truncation error in the numerical method, and round-off error in the computer. In order to have faith in the numerical solutions, we have to establish the order of magnitude of each of these errors and justify that they do not unduly influence our computed results.

1.1 Approximation of physical processes

There are many levels at which we can mathematically model the motion of a fluid. Choosing the appropriate model requires a trade-off between fidelity of the approximation and expense in obtaining the solution. If the world consisted of terra-flop personal computers, perhaps the Navier–Stokes equations would be used for all CFD calculations. Unfortunately, computer resources are limited today, even with the vast improvements in speed and memory of recent computers. Thus, one of the first tasks of the numerical analyst is to select a mathematical model of the fluid system which includes the important physical processes while being computationally affordable. Fortunately there is a clear hierarchy of physical models to choose from.

The most general model under routine use is at the level of the fluid molecule where the motion of individual molecules is tracked and inter-molecular interactions are simulated. For example, this level of approximation is required for the rarefied gases encountered during the reentry of space-craft in the upper atmosphere. Although this model can certainly be used at lower speeds and altitudes, it becomes prohibitively expensive to track individual molecules under non-rarefied conditions. Thus, another mathematical model is needed.

By far, the most commonly used model for fluid dynamics is the Navier–Stokes equations. Unlike the molecular model described above, the fluid is treated as a continuum and only statistical quantities such as temperature, bulk velocity, and pressure are available at this level of approximation. Fortunately, the Navier–Stokes equations accurately predict the dynamics of most common fluids under a wide range of conditions (the most notable exceptions are non-Newtonian fluids, rarefied gases, and flows with very strong shock waves). For low speed problems or liquid flows, we can make the further assumption that the density is approximately constant which leads to the somewhat simpler incompressible Navier–Stokes equations. We will have much more to say about the Navier–Stokes equations in later chapters.

In moving from the molecular description to the continuum model we basically performed an averaging process over the molecules to obtain bulk quantities such as temperature and pressure. It turns out that averaging (or more generally filtering) is one of the primary means of simplifying our mathematical model. For example, if we average the Navier–Stokes equations in one space dimension, then we are left with the two-dimensional Navier–Stokes equations. As long as the process that we wish to simulate is approximately two-dimensional then this will be an adequate model. Of course, we can continue by averaging over two spatial dimensions or even over all three directions if we are only interested in the variation of mean quantities.

If instead of averaging, we use a low-pass filter in space, then we obtain the so-called Large

Eddy Simulation (LES) equations. This model is very useful for turbulent flows which generally have a wide range of spatial length scales. By using a low-pass filter we only include the large scales, or eddies, in our computation while the small-scale eddies are modeled using empirical relationships. It turns out that the large eddies contain most of the energy, while the small eddies are nearly isotropic and therefore easier to model. By only explicitly including the large eddies in the computation, a tremendous savings is obtained over a full Navier–Stokes solution (often called Direct Numerical Simulation) which requires the resolution of all the scales of a turbulent flow.

At the next level in our hierarchy of models, we can average the Navier–Stokes equations in time (or use an ensemble average for unsteady flows) which leads to the Reynolds Averaged Navier–Stokes (RANS) equations. In this case, we are faced with the task of modeling all the scales of a turbulent flow. This task is much more difficult than that encountered with the LES equations. The development of accurate and robust turbulence models is an active area of research. This research is particularly important, since RANS solutions are currently the highest level of approximation commonly used in CFD calculations for engineering design.

Given the hierarchy of mathematical models that we have introduced, it is possible, under certain circumstances, to make further approximations that take into account special physical characteristics of the flow under consideration. For example, Prandlt's landmark discovery that viscous effects are primarily limited to a boundary layer near a solid surface has lead to the boundary layer equations which are a special form of the Navier–Stokes equations that are considerably easier to solve numerically. Outside of the boundary layer, which means most of the flow in the case of an aircraft, the flow is generally inviscid and the viscous terms in the Navier–Stokes equations can be dropped leading to the Euler equations. If there are no shock waves in the flow, then further simplification can be obtained by using the potential flow equations.

The admittedly brief discussion given here is only meant to give the reader an appreciation for the variety of mathematical models available in CFD and the importance in selecting the appropriate model before attempting a numerical solution. The most ingenious numerical method applied to an inappropriate flow model will result in, at best, an inefficient solution method, and at worst an unusable method.

1.2 Numerical approximation

Once an appropriate mathematical model is chosen the next step is to design a numerical method which obtains the solution with sufficient efficiency and accuracy for the problem at hand. There are three primary techniques used to convert the partial differential equations which form our mathematical model into a set of algebraic equations that can be solved on a computer: finite difference, finite volume, and finite element.[1] Each of these methods involve dividing or discretizing space and time in some manner.

The most basic method, and the one that we will discuss in the most detail, is the finite difference method. Here the equations of motion are discretized on a series of node points and the differential operators are approximated using algebraic expressions derived from Taylor series expansions. Finite difference methods have the advantage of being simple to understand and code on a computer and it is relatively easy to implement highly accurate finite difference algorithms. Their main disadvantage is that they generally require the domain to be divided into a rectangular array of node points (called a structured mesh) which can make implementation difficult for complex geometries.

In the finite volume method, the equations are written in integral form and the domain is divided into small control volumes on which the integral equations are enforced using numerical integration. The finite volume method has the advantage that the control volumes can be of arbitrary shape with triangles commonly used in two-dimensions and tetrahedra in three dimensions. This gives the finite volume method extensive flexibility in representing complex domains through the use of unstructured meshes. One challenge in finite volume research is in the development of highly accurate algorithms.

Similar to finite volumes, the finite element method divides the domain into volumes, called elements, and over each element the approximate solution is represented by a set of interpolating functions. A significant advantage of the finite element method is that formal mathematical analysis can be used to prove accuracy and convergence under some conditions. Finite elements can very naturally be used on unstructured meshes and high accuracy can be obtained by judicious selection of the interpolating functions over each element. The main disadvantage of the finite element method lies in the complexity of the computer algorithms and the reduction in computational efficiency compared to finite volume and finite difference. With improvements in computer capability and the need to perform calculations on more complex flow systems, the finite element method has become increasingly popular in recent years.

All of these methods share similarities, and under certain (rather limited) circumstances, each can be shown to be equivalent to the other. That is not to say that one method does not have advantages over the other for certain problems. As indicated above, finite differences are noted

[1]A fourth technique, called spectral methods, based on global function approximation is often used for calculations requiring extremely high accuracy. However, we will not have the opportunity to discuss these methods in this class.

for their simplicity and computational efficiency, finite volumes for their ability to handle complex geometries, and finite elements because of their high accuracy and mathematical foundation. In this class, we will focus primarily on finite difference methods since this technique is the most established in current CFD codes and since numerical analysis applied to finite difference methods are also useful for analyzing finite volume and finite element methods.

1.3 Sources of error

All numerical solutions have sources of error. It is our responsibility to ensure that these errors are small enough so that they do not render our solution meaningless. The first possible source of error comes in the selection of our mathematical approximation. If we assume that the flow is two-dimensional when three-dimensional effects are important then an error has been made which may render our solutions useless. A more subtle error occurs in LES and RANS calculations. Here we rely on an empirical model to accurately predict the dynamics of the unresolved turbulence. Clearly errors in the predicted solutions will occur and they are a function of both the turbulence model used and the flow to which they are applied. A careful selection and validation of the mathematical model is our primary weapon against these types of errors.

Any numerical algorithm, be it finite difference or finite element, introduces numerical errors into the solution. These errors are commonly called truncation error in finite difference analysis since they originate from the truncation of a Taylor series expansion of the solution. Other sources of numerical errors in CFD include the treatment of boundary conditions and non-uniform meshes. We will have much more to say about numerical errors in later chapters.

Once the mathematical model is selected and an appropriate numerical method is implemented there is one additional source of error. These errors come from the fact that computers only contain a finite precision approximation to floating point numbers. Because of this, every floating point operation (flop) introduces a small amount of error due to round-off of the least significant figure. Under most circumstances, this round-off error is sufficiently small that it does not adversely effect the solution. However, it is possible for round-off error to have a catastrophic effect on the solution for poorly written algorithms. A primary source of catastrophic round-off error occurs when two numbers that are very nearly equal are subtracted and you must be careful to ensure that your algorithms are written to avoid this occurrence. Some problems in CFD, such as instability waves in a laminar boundary layer and aeroacoustics, are particularly sensitive to round-off errors and extreme care must be used in solving these types of problems. Students interested in learning more about finite precision calculations should refer to one of the many texts on numerical computation.[2]

[2]W. Cheney & D. Kincad, Numerical Mathematics and Computing, 2nd edition, Brooks/Cole, 1985.

Chapter 2

Interpolation

A fundamental problem of numerical analysis rests in the approximation of a function by a discrete set of data. The data points might represent the nodes in a finite difference computation or may be the result of an experimental measurement. Whichever the case, we are faced with the task of fitting a smooth curve through the data that approximates, in some sense, the actual function so that we can estimate its value between any two data points or estimate the derivatives or integral of the function. Here, we will discuss *interpolation* which refers to the case where a smooth curve is required to pass through all the data points. If the data has some uncertainty (which is often the case for experiments) then the method of *least squares*, which does not strictly require the approximating function to pass through the data points, may be more appropriate.

2.1 Polynomial Interpolation

Given a set of $n+1$ (possibly non-equally spaced) data (x_i, y_i), we can construct a polynomial of degree n which passes through the data

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \tag{2.1}$$

This leads to a system of $n+1$ equations in $n+1$ unknowns, $a_0, a_1, \cdots, a_n$ of the form

$$y_i = P(x_i) = a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_n x_i^n \qquad i = 0, \cdots, n \tag{2.2}$$

This linear system of equations can be solved by Gaussian elimination to determine the unknown coefficients. Unfortunately this procedure is not very useful since the system of equations generally becomes ill-conditioned for large n. Fortunately, there is an alternative method for defining the polynomial which leads to an explicit (you don't have to solve a system of equations) way of determining the coefficients.

Let's define a polynomial of degree n which is associated with each point x_j

$$L_j(x) = \alpha_j (x - x_0)(x - x_1) \cdots (x - x_{j-1})(x - x_{j+1}) \cdots (x - x_n) \tag{2.3}$$

where α_j is a constant to be determined. Using the product notation for L_j leads to

$$L_j(x) = \alpha_j \prod_{\substack{i=0 \\ i \neq j}}^{n} (x - x_i) \tag{2.4}$$

Note that by construction $L_j(x_i) = 0$ for $i \neq j$ and that

$$L_j(x_j) = \alpha_j \prod_{\substack{i=0 \\ i \neq j}}^{n} (x_j - x_i) \tag{2.5}$$

If we let

$$\alpha_j = \left[\prod_{\substack{i=0 \\ i \neq j}}^{n} (x_j - x_i) \right]^{-1} \tag{2.6}$$

then

$$L_j(x) = \begin{cases} 0 & \text{if } x \neq x_j \\ 1 & \text{if } x = x_j \end{cases} \tag{2.7}$$

Forming the linear combination:

$$P(x) = \sum_{j=0}^{n} y_j L_j(x) \tag{2.8}$$

leads to a polynomial of degree n since it is composed of a linear combination of polynomials, each of degree n. This polynomial form is called a Lagrange polynomial and it is a classical way of representing a polynomial interpolating function in a convenient and efficient manner. The Lagrange polynomial is constructed to pass identically through the data points. For example, at $x = x_i$

$$P(x_i) = y_0 L_0(x_i) + y_1 L_1(x_i) + \cdots + y_i L_i(x_i) + \cdots + y_n L_n(x_i) \tag{2.9}$$

which by (2.7) simplifies to

$$P(x_i) = y_i \tag{2.10}$$

Thus, the polynomial P is the desired interpolate to the set of data points. It can be readily shown that polynomial interpolation is unique so that there is only one polynomial of degree n which passes through a set of $n + 1$ data points. The Lagrange polynomial is just a more convenient and numerically more attractive way of expressing the same polynomial that we would obtain by solving a system of equations like (2.2).

Great caution must be used when performing polynomial interpolation of large data sets, say greater than 10 points. Although the interpolant will pass through the data points it can oscillate wildly between them, leading to large errors in the interpolated values and especially derivatives. Better results can be obtained by using piecewise Lagrange interpolation. Instead of fitting a single polynomial of order n to all the data, the dataset is broken into segments (or elements) with lower order interpolation used on each segment. This approach is widely used and is the basis for the Finite Element Method. A limitation of this technique is that derivatives of the interpolant are discontinuous at the segment boundaries. To address this limitation, we now introduce spline interpolation.

2.2 Spline Interpolation

A spline is a function which consists of piecewise polynomials which are joined together with certain smoothness conditions. The simplest example (a spline of degree 1) is the piecewise linear polynomial which consists of linear segments that are connected to achieve a continuous function. With each increase in polynomial order, we can enforce continuity of higher and higher derivatives. By far the most popular spline function is the cubic spline for which the first two derivatives are continuous at the data points (often called *knots* in the theory of splines). Interpolation with cubic splines is essentially equivalent to passing a flexible plastic ruler through the dataset.

Let $g_i(x)$ be a cubic polynomial in the interval $x_i \leq x \leq x_{i+1}$ and let $g(x)$ denote the collection of all the cubics for the entire range of x. Since $g(x)$ is piecewise cubic, its second derivative, g'', is piecewise linear. For an arbitrary interval, $x_i \leq x \leq x_{i+1}$, the equation for the second derivative can be written as

$$g_i''(x) = g''(x_i)\frac{x - x_{i+1}}{x_i - x_{i+1}} + g''(x_{i+1})\frac{x - x_i}{x_{i+1} - x_i} \tag{2.11}$$

Note that in (2.11) continuity of the second derivative has been enforced at the data points. Integrating this equation twice leads to

$$g_i'(x) = \frac{g''(x_i)}{x_i - x_{i+1}}\frac{(x - x_{i+1})^2}{2} + \frac{g''(x_{i+1})}{x_{i+1} - x_i}\frac{(x - x_i)^2}{2} + C_1 \tag{2.12}$$

and

$$g_i(x) = \frac{g''(x_i)}{x_i - x_{i+1}}\frac{(x - x_{i+1})^3}{6} + \frac{g''(x_{i+1})}{x_{i+1} - x_i}\frac{(x - x_i)^3}{6} + C_1 x + C_2 \tag{2.13}$$

The constants C_1 and C_2 are determined by constraining the interpolant to pass through the data points

$$g_i(x_i) = y_i \qquad g_i(x_{i+1}) = y_{i+1} \tag{2.14}$$

Using these values of C_1 and C_2 in (2.13) leads to the spline interpolant

$$\begin{aligned} g_i(x) &= \frac{g''(x_i)}{6}\left[\frac{(x_{i+1}-x)^3}{\Delta_i} - \Delta_i(x_{i+1}-x)\right] + \\ & \frac{g''(x_{i+1})}{6}\left[\frac{(x-x_i)^3}{\Delta_i} - \Delta_i(x-x_i)\right] + y_i\frac{x_{i+1}-x}{\Delta_i} + y_{i+1}\frac{x-x_i}{\Delta_i} \end{aligned} \tag{2.15}$$

where $x_i \le x \le x_{i+1}$ and $\Delta_i = x_{i+1} - x_i$. So far we have enforced continuity of the second derivative in (2.11) and of the function in (2.14). It remains to enforce continuity of the first derivative at the data points

$$g_i'(x_i) = g_{i-1}'(x_i) \tag{2.16}$$

which is accomplished by the proper choice of the values $g''(x_i)$ which are still unknown. The desired system of equations for $g''(x_i)$ is obtained by using (2.12) in (2.16):

$$\begin{aligned} &\frac{\Delta_{i-1}}{6}g''(x_{i-1}) + \frac{\Delta_{i-1}+\Delta_i}{3}g''(x_i) + \frac{\Delta_i}{6}g''(x_{i+1}) = \\ &\frac{y_{i+1}-y_i}{\Delta_i} - \frac{y_i - y_{i-1}}{\Delta_{i-1}} \qquad\qquad i = 1, 2, 3, \ldots, n-1 \end{aligned} \tag{2.17}$$

This is a system of $n-1$ equations for the $n+1$ unknowns $g''(x_0)$, $g''(x_1)$, ..., $g''(x_n)$. The equations are in tridiagonal form and are diagonally dominant so that they can be solved efficiently. The additional two equations required to solve the problem are determined by prescribing the end conditions. Some possible choices are

1. Parabolic run-out

$$g''(x_0) = g''(x_1) \tag{2.18a}$$

$$g''(x_n) = g''(x_{n-1}) \tag{2.18b}$$

2. Free run-out also called a natural spline

$$g''(x_0) = 0 \tag{2.19a}$$

$$g''(x_n) = 0 \tag{2.19b}$$

3. Combination of parabolic and free run-out

$$g''(x_0) = \alpha g''(x_1) \tag{2.20a}$$

$$g''(x_n) = \beta g''(x_{n-1}) \tag{2.20b}$$

where α and β are set by the user.

4. Periodic

$$g''(x_0) = g''(x_{n-1}) \tag{2.21a}$$

$$g''(x_1) = g''(x_n) \tag{2.21b}$$

The procedure used in spline interpolation is to solve the system of equations (2.17) along with the appropriate end conditions to determine the $g''(x_i)$. The result is used in equation (2.15) to determine the interpolant $g_i(x)$ for the interval $x_i \leq x \leq x_{i+1}$. In general, spline interpolation is preferable to Lagrange polynomial interpolation since it is efficient and leads to smooth curves.

Chapter 3

Numerical Differentiation

Constructing approximations to derivatives is an essential step toward obtaining numerical solutions to differential equations. Typically we think of numerical differentiation as a point operator which takes function values at neighboring points and combines them to approximate a derivative. This is the classical *Finite Difference* approach. However, numerical derivatives can also be easily computed using the polynomial interpolations presented in the previous chapter. Constructing derivative approximations based on interpolation is the heart of the *Finite Element* method. Although we will primarily focus on the finite difference approach, it is often possible to cast a finite difference point operator into an analogous polynomial interpolation method and, depending on the application, both frameworks are useful to the numerical analyst.

3.1 Finite Differences

We would like to approximate the derivative of a function which is defined on a discrete set of data points $x_0, x_1, \ldots, x_n$. This can be readily accomplished using finite difference formulas which are derived from Taylor series expansions. For example, the Taylor series expansion of the function $f(x)$ about the point x_j can be used to construct an approximation of the first derivative at x_j.

$$f(x_{j+1}) = f(x_j) + (x_{j+1} - x_j)f'(x_j) + \frac{(x_{j+1} - x_j)^2}{2!}f''(x_j) + \frac{(x_{j+1} - x_j)^3}{3!}f'''(x_j) + \cdots \tag{3.1}$$

Solving for the derivative, $f'(x_j)$, leads to

$$f'(x_j) = \frac{f(x_{j+1}) - f(x_j)}{\Delta x_j} - \frac{\Delta x_j}{2}f''(x_j) - \frac{(\Delta x_j)^2}{6}f''(x_j) + \cdots \tag{3.2}$$

where $\Delta x_j = x_{j+1} - x_j$ is the mesh size. As a shorthand notation, we also use h_j to indicate the mesh size. When there is no subscript on Δx or h the mesh spacing is uniform. Using the shorthand notation, we can rewrite this formula as

$$\frac{f_{j+1} - f_j}{h} - f'(x_j) = \frac{h}{2}f''(x_j) + \frac{h^2}{6}f'''(x_j) + \cdots \tag{3.3}$$

In this form it is clear that the discrete terms on the left side of the equation represent the first derivative with a certain amount of error which is given by the terms on the right side of the equals sign. This error, called truncation error, depends on the mesh size. The order of accuracy of the method is given by the error term which contains the lowest power of the step size. In this case the order of accuracy is one which means that if the mesh size is reduced by a factor of 2 then the error is also, approximately, reduced by a factor of 2. This finite difference formula can be written in the summary form

$$f'_j = \frac{f_{j+1} - f_j}{h} + O(h) \tag{3.4}$$

which is referred to as the first-order forward difference. In a similar manner we can derive the first derivative approximation

$$f'_j = \frac{f_j - f_{j-1}}{h} + O(h) \tag{3.5}$$

which is called the first-order backward difference. Higher order schemes, which give greater accuracy, can be derived by using Taylor series expansions of the function f at other locations. For example, the central difference formula can be obtained by subtracting the following Taylor series expansions:

$$f_{j+1} = f_j + hf'_j + \frac{h^2}{2}f''_j + \frac{h^3}{6}f'''_j + \cdots \tag{3.6}$$

$$f_{j-1} = f_j - hf'_j + \frac{h^2}{2}f''_j - \frac{h^3}{6}f'''_j + \cdots \tag{3.7}$$

which leads to the second-order formula

$$f'_j = \frac{f_{j+1} - f_{j-1}}{2h} - \frac{h^2}{3}f'''_j + \cdots \tag{3.8}$$

This process can be extended to derive higher-order formulae such as the fourth-order accurate expression

$$f'_j = \frac{f_{j-2} - 8f_{j-1} + 8f_{j+1} - f_{j+2}}{12h} + O(h^4) \tag{3.9}$$

Formulae such as these are easily derived using the general procedure called a Taylor table. Suppose that we want to construct the most accurate difference scheme that uses function values at j, $j+1$, and $j+2$. Thus we want an expression of the form

$$f'_j + \frac{1}{h}\sum_{k=0}^{2} a_k f_{j+k} = O(h^?) \tag{3.10}$$

where a_k are the coefficients that result from the linear combination of Taylor series expansions. It is convenient to form the linear combination using the following Taylor table

	f_j	f'_j	f''_j	f'''_j
hf'_j	0	h	0	0
$a_0 f_j$	a_0	0	0	0
$a_1 f_{j+1}$	a_1	$a_1 h$	$a_1 \frac{h^2}{2}$	$a_1 \frac{h^3}{6}$
$a_2 f_{j+2}$	a_2	$2a_2 h$	$a_2 \frac{(2h)^2}{2}$	$a_2 \frac{(2h)^3}{6}$

By summing the columns in the Taylor table we obtain the sum indicated in equation (3.10), multiplied through by h:

$$\begin{aligned} hf'_j + \sum_{k=0}^{2} a_k f_{j+k} &= (a_0 + a_1 + a_2) f_j + (1 + a_1 + 2a_2) h f'_j + \\ & \quad (\frac{a_1}{2} + 2a_2) h^2 f''_j + (\frac{a_1}{6} + \frac{4a_2}{3}) h^3 f'''_j + \cdots \end{aligned} \tag{3.11}$$

To get the highest accuracy requires that we set as many of the low-order terms to zero as is possible. Since we have three coefficients, we can set the coefficients of the first three terms to zero:

$$a_0 + a_1 + a_2 = 0 \tag{3.12a}$$

$$a_1 + 2a_2 = -1 \tag{3.12b}$$

$$a_1/2 + 2a_2 = 0 \tag{3.12c}$$

The solution to these equations is

$$a_0 = \frac{3}{2} \qquad a_1 = -2 \qquad a_2 = \frac{1}{2} \tag{3.13}$$

which results in the second-order finite difference

$$f'_j = \frac{-3f_j + 4f_{j+1} - f_{j+2}}{2h} + \frac{h^2}{3} f'''_j + \cdots \tag{3.14}$$

The proceeding discussion has been limited to first derivative approximations, however, the same technique can be applied for second and higher derivatives. Consider the second derivative approximation using the points $j-1, j, j+1$. This leads to an approximation of the form

$$f_j'' + \frac{1}{h^2}\sum_{k=0}^{2} a_k f_{j-1+k} = O(h^?) \tag{3.15}$$

The Taylor table for this case is given by

	f_j	f_j'	f_j''	f_j'''	$f_j^{(iv)}$
$h^2 f_j''$	0	0	h^2	0	0
$a_0 f_{j-1}$	a_0	$-a_0 h$	$a_0\frac{h^2}{2}$	$-a_0\frac{h^3}{6}$	$a_0\frac{h^4}{24}$
$a_1 f_j$	a_1	0	0	0	0
$a_2 f_{j+1}$	a_2	$a_2 h$	$a_2\frac{h^2}{2}$	$a_2\frac{h^3}{6}$	$a_2\frac{h^4}{24}$

Requiring that the column sums are zero leads to the following equations

$$a_0 + a_1 + a_2 = 0 \tag{3.16a}$$

$$-a_0 + a_2 = 0 \tag{3.16b}$$

$$a_0 + a_2 = -2 \tag{3.16c}$$

which has the solution

$$a_0 = a_2 = -1 \qquad a_1 = 2 \tag{3.17}$$

Due to symmetry, the fourth column also sums to zero (in fact all odd derivative terms will be zero) so that the second derivative approximation given by

$$f_j'' = \frac{f_{j-1} - 2f_j + f_{j+1}}{h^2} + \frac{h^2}{12} f_j^{(iv)} + \cdots \tag{3.18}$$

is second-order accurate.

3.2 Polynomial Interpolation and Finite Differences

In Chapter 2 we derived the following expression for the Lagrangian interpolation polynomial of degree n

$$f(x) = \sum_{j=0}^{n} f_j L_j(x) \tag{3.19}$$

If, for example, we consider the Lagrangian polynomial for quadratic interpolation then

$$\begin{aligned} f(x) &= f_{j-1}\frac{(x-x_j)(x-x_{j+1})}{(x_{j-1}-x_j)(x_{j-1}-x_{j+1})} + f_j\frac{(x-x_{j-1})(x-x_{j+1})}{(x_j-x_{j-1})(x_j-x_{j+1})} + \\ & f_{j+1}\frac{(x-x_{j-1})(x-x_j)}{(x_{j+1}-x_{j-1})(x_{j+1}-x_j)} \end{aligned} \tag{3.20}$$

If the mesh is uniform this can be rewritten as

$$\begin{aligned} f(x) &= f_{j-1}\frac{(x-x_j)(x-x_{j+1})}{2h^2} - f_j\frac{(x-x_{j-1})(x-x_{j+1})}{h^2} + \\ & f_{j+1}\frac{(x-x_{j-1})(x-x_j)}{2h^2} \end{aligned} \tag{3.21}$$

Taking the derivative of this expression and evaluating at x_j yields

$$f'_j = \frac{f_{j+1}-f_{j-1}}{2h} \tag{3.22}$$

which is identical to the second-order central difference formula that we derived using Taylor series expansions. Similarly, the second derivative is given by

$$f''_j = \frac{f_{j+1}-2f_j+f_{j-1}}{h^2} \tag{3.23}$$

which is again the classic central difference expression for the second derivative at x_j. Finite difference schemes that can be derived from the polynomial form (3.19) are called Lagrangian approximations. It is often a very useful and powerful concept to think of finite difference schemes in terms of their implied interpolation formula.

3.3 Padé Approximations

The procedure that we developed for obtaining finite difference formula using Taylor series can be generalized to also include the derivatives at the same points. For example, if in addition to f'_j, f_{j-1}, f_j, f_{j+1} we also include f'_{j-1} and f'_{j+1} then we have the following expression for the derivative

$$b_0 f'_{j-1} + f'_j + b_1 f'_{j+1} + \frac{1}{h}\sum_{k=0}^{2} a_k f_{j-1+k} = O(h^?) \tag{3.24}$$

and we can construct the following Taylor table

	f_j	f'_j	f''_j	f'''_j	$f_j^{(iv)}$	$f_j^{(v)}$
hf'_{j-1}	0	b_0h	$-b_0h^2$	$b_0\frac{h^3}{2}$	$-b_0\frac{h^4}{6}$	$b_0\frac{h^5}{24}$
hf'_j	0	1	0	0	0	0
hf'_{j+1}	0	b_1h	b_1h^2	$b_1\frac{h^3}{2}$	$b_1\frac{h^4}{6}$	$b_1\frac{h^5}{24}$
a_0f_{j-1}	a_0	$-a_0h$	$a_0\frac{h^2}{2}$	$-a_0\frac{h^3}{6}$	$a_0\frac{h^4}{24}$	$-a_0\frac{h^5}{120}$
a_1f_j	a_1	0	0	0	0	0
a_2f_{j+1}	a_2	a_2h	$a_2\frac{h^2}{2}$	$a_2\frac{h^3}{6}$	$a_2\frac{h^4}{24}$	$a_2\frac{h^5}{120}$

Summing the columns of this table and setting equal to zero leads to the following system of equations

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & -2 & 2 \\ -1 & 0 & 1 & 3 & 3 \\ 1 & 0 & 1 & -4 & 4 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ b_0 \\ b_1 \end{Bmatrix} = \begin{Bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \tag{3.25}$$

which has the solution $[a_0, a_1, a_2, b_0, b_1] = \frac{1}{4}[3, 0, -3, 1, 1]$. The method can be expressed as

$$f'_{j-1} + 4f'_j + f'_{j+1} = \frac{3}{h}(f_{j+1} - f_{j-1}) + \frac{h^4}{30}f_j^{(v)} + \cdots \tag{3.26}$$

$$j = 1, 2, 3, \ldots, n-1 \tag{3.27}$$

This expression for the derivative is fundamentally different from the previous expressions we have derived. In equation (3.8) the first derivative is *explicitly* determined by neighboring function values. However, in equation (3.26), the derivative at j depends *implicitly* on the derivative values at $j-1$ and $j+1$ in addition to the neighboring function values. To compute the derivative from equation (3.26) requires us to solve a tridiagonal system of equations. Since we only have equations for the interior points, special treatment is required near the boundaries. Usually a lower order one-sided difference is used to approximate f'_0 and f'_n.

By including the derivative values in addition to the function values, we have increased the order of accuracy from second to fourth-order. Despite this increase, the scheme is still *compact* in that it only requires information from the neighboring points, $j-1$ and $j+1$, although the scheme is global in the sense that all the functional values are required to obtain the derivative at a given point. The price that we pay for the increased accuracy is the need to solve a tridiagonal system of equations. However, since simple and efficient algorithms are available to solve such systems, the Padé difference technique has become very popular for applications requiring extremely high accuracy such as transitional and turbulent flows.

Just as the explicit finite difference schemes where shown to be equivalent to Lagrangian interpolation, the Padé differencing schemes are equivalent to Hermitian interpolation. To construct a polynomial for $f(x)$, Hermitian interpolation uses both the values of the function and its derivatives. For example,

$$f(x) = \sum_{j=0}^{n} f_j L_j(x) + \sum_{j=0}^{n} f'_j H_j(x) \tag{3.28}$$

is a Hermitian interpolation using the first derivatives. Higher order derivatives can also be included depending on the application. The family of Hermitian polynomials includes cubic spline approximations as discussed in Section 2.2. In fact, the first derivative expression (3.26), derived above, is equivalent to evaluating the first derivative using a cubic spline interpolation. Additional details on Hermitian polynomials and higher-order splines can be found in many texts on numerical methods.

Chapter 4

Numerical Integration

The goal of numerical integration or quadrature is to estimate the integral of the function f in the interval $[a, b]$.

$$I = \int_a^b f(x)dx \tag{4.1}$$

The function may be defined analytically or only on a set of discrete data points $x_0 = a$, x_1, x_2, ..., $x_n = b$. Whichever the case, we wish to use the discrete data to convert the integral into a finite sum of the form

$$I = \int_a^b f(x)dx \approx \sum_{j=0}^{n} w_j f(x_j) \tag{4.2}$$

The choice of weights, w_j, will influence the accuracy of the approximation.

Two popular choices for numerical integration are trapezoidal rule and Simpson's rule. For one interval the trapezoidal rule is given by

$$\int_{x_j}^{x_{j+1}} f(x)dx = \frac{\Delta x}{2}(f_i + f_{i+1}) \tag{4.3}$$

and for the entire interval, the trapezoidal rule becomes

$$I \approx \Delta x \left(\frac{f_0}{2} + \frac{f_n}{2} + \sum_{j=1}^{n-1} f_j \right) \tag{4.4}$$

where $\Delta x = h = x_{j+1} - x_j$ and uniform spacing has been assumed. A more accurate method of approximating the integral is Simpson's Rule which is given by

$$I \approx \frac{\Delta x}{3} \left(f_0 + f_n + 4 \sum_{\substack{j=1 \\ j=odd}}^{n-1} f_j + 2 \sum_{\substack{j=2 \\ j=even}}^{n-2} f_j \right) \tag{4.5}$$

Note that Simpson's rule requires that the number of points, $(n + 1)$, be odd.

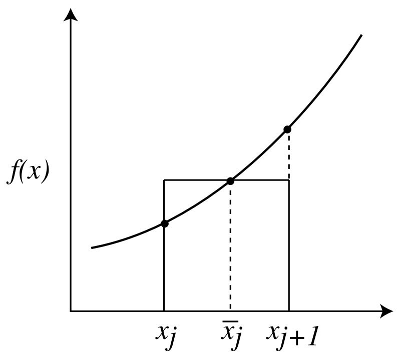

Figure 4.1: Midpoint rule

4.1 Analysis of Integration Errors

As in the case of interpolation and differentiation, we are interested in determining the error, or more precisely, the order of accuracy of integration scheme.

4.1.1 Midpoint Rule

Let's begin with the midpoint (or rectangle) rule for the interval $[x_j, x_{j+1}]$

$$\int_{x_j}^{x_{j+1}} f(x)dx \approx h_j f(\bar{x}_j) \tag{4.6}$$

where $\bar{x}_j = (x_j + x_{j+1})/2$ is the midpoint of the interval. We can replace the integrand with its Taylor series expansion about $\bar{x}_j$

$$f(x) = f(\bar{x}_j) + (x - \bar{x}_j)f'(\bar{x}_j) + \frac{(x-\bar{x}_j)^2}{2}f''(\bar{x}_j) + \frac{(x-\bar{x}_j)^3}{6}f'''(\bar{x}_j) + \dots \tag{4.7}$$

yielding

$$\int_{x_j}^{x_{j+1}} f(x)dx = h_j f(\bar{x}_j) + \frac{1}{2}(x-\bar{x}_j)^2\Big|_{x_j}^{x_{j+1}} f'(\bar{x}_j) + \frac{1}{6}(x-\bar{x}_j)^3\Big|_{x_j}^{x_{j+1}} f''(\bar{x}_j) + \dots \tag{4.8}$$

Terms with even powers of $(x - \bar{x}_j)$ vanish due to symmetry so that we are left with

$$\int_{x_j}^{x_{j+1}} f(x)dx = h_j f(\bar{x}_j) + \frac{h_j^3}{24}f''(\bar{x}_j) + \frac{h_j^5}{1920}f^{(iv)}(\bar{x}_j) + \dots \tag{4.9}$$

Thus, midpoint rule is third order accurate for one interval.

To obtain the integral for the entire domain we sum both sides of equation (4.9) assuming uniform spacing

$$I = \sum_{j=0}^{n-1} \int_{x_j}^{x_{j+1}} f(x)dx = h \sum_{j=0}^{n-1} f(\bar{x}_j) + \frac{h^3}{24} \sum_{j=0}^{n-1} f''(\bar{x}_j) + \dots \tag{4.10}$$

The Mean Value Theorem applied to summations says that there exists a point, ξ, in the interval $[a, b]$, such that

$$\sum_{j=0}^{n-1} f''(\bar{x}_j) = nf''(\xi) \tag{4.11}$$

Since $n = (b - a)/h$ we obtain

$$I = h \sum_{j=0}^{n-1} f(\bar{x}_j) + (b - a) \frac{h^2}{24} f''(\xi) + \dots \tag{4.12}$$

Thus, we conclude that midpoint rule is second order accurate over the entire interval.

4.1.2 Trapezoidal Rule

We can perform a similar analysis for the trapezoidal rule starting from equation (4.9). First let's form Taylor series expansions for $f(x_j)$ and $f(x_{j+1})$ about the midpoint $\bar{x}_j$.

$$f(x_j) = f(\bar{x}_j) - \frac{h_j}{2} f'(\bar{x}_j) + \frac{h_j^2}{8} f''(\bar{x}_j) - \frac{h_j^3}{48} f'''(\bar{x}_j) + \cdots \tag{4.13}$$

$$f(x_{j+1}) = f(\bar{x}_j) + \frac{h_j}{2} f'(\bar{x}_j) + \frac{h_j^2}{8} f''(\bar{x}_j) + \frac{h_j^3}{48} f'''(\bar{x}_j) + \cdots \tag{4.14}$$

Adding these two expansions and dividing by two yields

$$\frac{f(x_j) + f(x_{j+1})}{2} = f(\bar{x}_j) + \frac{h_j^2}{8} f''(\bar{x}_j) + \frac{h_j^4}{384} f^{(iv)}(\bar{x}_j) + \cdots \tag{4.15}$$

Solving for $f(\bar{x}_j)$ and substituting into equation (4.9) we obtain

$$\int_{x_j}^{x_{j+1}} f(x)dx = h_j \frac{f(x_j) + f(x_{j+1})}{2} - \frac{1}{12} h_j^3 f''(\bar{x}_j) - \frac{1}{480} h_j^5 f^{(iv)}(\bar{x}_j) + \cdots \tag{4.16}$$

which demonstrates that trapezoidal rule is also third order accurate over a single interval. Note that the coefficient of the leading order term is twice in magnitude but of opposite sign compared to the midpoint rule. Assuming a uniform mesh and using the mean value theorem we find that over the entire domain, the trapezoidal rule leads to

$$I = \frac{h}{2}\left(f(a) + f(b) + 2\sum_{j=1}^{n-1} f(x_j)\right) - (b-a)\frac{h^2}{12}f''(\xi) + \ldots \tag{4.17}$$

Thus, the trapezoidal rule is also second order accurate over the entire domain.

4.1.3 Trapezoidal Rule with End-Corrections

If we know the derivatives of the integrand at the end points, it is possible to increase the accuracy of the trapezoidal rule from second order to fourth order. This is accomplished by using the second order finite difference formula for $f''(\bar{x}_j)$ in equation (4.16).

$$I_j = h_j\frac{f_j + f_{j+1}}{2} - \frac{h_j^3}{12}\frac{f'_{j+1} - f'_j}{h_j} + O(h_j^5) \tag{4.18}$$

If we consider a uniform mesh then using the mean value theorem

$$I = \frac{h}{2}\sum_{j=0}^{n-1}(f_j + f_{j+1}) - \frac{h^2}{12}\sum_{j=0}^{n-1}\left(f'_{j+1} - f'_j\right) + O(h^4) \tag{4.19}$$

All terms in the second summation cancel except the end points leaving

$$I = \frac{h}{2}\sum_{j=0}^{n-1}(f_j + f_{j+1}) - \frac{h^2}{12}\left(f'(b) - f'(a)\right) + O(h^4) \tag{4.20}$$

As advertised, trapezoidal rule with end-correction is fourth order accurate as long as the derivatives at the end points are known.

4.1.4 Simpson's Rule

One interval Simpson's Rule covers $[x_j, x_{j+2}]$ with the midpoint at x_{j+1}. Simpson's formula in equation (4.5) can be written as

$$S(f) = \frac{2}{3}M(f) + \frac{1}{3}T(f) \tag{4.21}$$

where $M(f)$ is the midpoint rule and $T(f)$ is the trapezoidal rule. Combining equations (4.9) and (4.16) according to this formula leads to an expression that is fifth order accurate over a single interval and thus fourth order accurate for the whole domain.

4.2 Richardson Extrapolation and Romberg Integration

The basic ideal of Richardson extrapolation is that a more accurate numerical solution can be obtained by combining less accurate solutions if you know the form of the truncation error. In premise, this method can be applied to any numerical method: integration, differentiation, interpolation, etc. As a particular example, we will use Richardson extrapolation to improve the accuracy of the integral

$$I = \int_a^b f(x)dx \tag{4.22}$$

evaluated using the trapezoidal rule.

From the error analysis of the trapezoidal rule we have the following expression for the integral

$$I = \frac{h}{2}\left[f(a) + f(b) + 2\sum_{j=1}^{n-1} f_j\right] + c_1h^2 + c_2h^4 + c_3h^6 + \ldots \tag{4.23}$$

If we let

$$\tilde{I}_1 = \frac{h}{2}\left[f(a) + f(b) + 2\sum_{j=1}^{n-1} f_j\right] \tag{4.24}$$

then the trapezoidal rule can be rewritten as

$$\tilde{I}_1 = I - c_1h^2 - c_2h^4 - c_3h^6 - \ldots \tag{4.25}$$

If we evaluate the integral with half the step size $h_2 = h/2$ then

$$\tilde{I}_2 = I - c_1\frac{h^2}{4} - c_2\frac{h^4}{16} - c_3\frac{h^6}{64} - \ldots \tag{4.26}$$

By taking a linear combination of I_1 and I_2, we can eliminate the $O(h^2)$ terms

$$\frac{4\tilde{I}_2 - \tilde{I}_1}{3} = I + \frac{1}{4}c_2h^4 + \frac{5}{16}c_3h^6 + \ldots \tag{4.27}$$

which yields a fourth order approximation for I. This procedure is the essence of Richardson extrapolation – two estimates of I have been used to obtain a more accurate estimate. We can continue by evaluating I using $h_3 = h/4$ to give

$$\tilde{I}_3 = I - c_1\frac{h^2}{16} - c_2\frac{h^4}{256} - c_3\frac{h^6}{4096} - \ldots \tag{4.28}$$

Taking a linear combination of I_2 and I_3 to remove the $O(h^2)$ terms yields

$$\frac{4\tilde{I}_3 - \tilde{I}_2}{3} = I + \frac{1}{64}c_2h^4 + \frac{5}{1024}c_3h^6 + \ldots \tag{4.29}$$

A linear combination of equations (4.27) and (4.29) can be formed to eliminate the $O(h^4)$ terms resulting is a sixth order accurate estimate. The procedure can be continued indefinitely and when applied to the trapezoidal rule, this algorithm is called Romberg integration. The following diagram summaries the process of Richardson extrapolation.

$$\begin{tikzcd}
O(h^2) & O(h^4) & O(h^6) \\
\tilde{I}_1 \arrow[dr] & & \\
\tilde{I}_2 \arrow[r] \arrow[dr] & \frac{4\tilde{I}_2 - \tilde{I}_1}{3} \arrow[dr] & \\
\tilde{I}_3 \arrow[r] & \frac{4\tilde{I}_3 - \tilde{I}_2}{3} \arrow[r] & {}
\end{tikzcd}$$

4.3 Adaptive Quadrature

If the function to be integrated varies rapidly in some regions and slowly in others, the use of a uniform mesh size over the domain can be quite inefficient. Adaptive quadrature methods automatically select the mesh size so that the result meets some user prescribed accuracy requirement. Mathematically, we say that for a minimum number of function evaluations we would like a numerical estimate, $\tilde{I}$, of the integral such that

$$\left| \tilde{I} - \int_a^b f(x)dx \right| \leq \epsilon \tag{4.30}$$

where ϵ is a user supplied error tolerance.

As an example, consider Simpson's rule as the base method over the domain $[a, b]$ which is divided into intervals $[x_j, x_{j+1}]$. Dividing this interval into two panels and using Simpson's rule leads to

$$S_j^{(1)} = \frac{h_j}{6}\left[f(x_j) + 4f\left(x_j + \frac{h_j}{2}\right) + f(x_j + h_j) \right] \tag{4.31}$$

If we further divide the interval into four panels then we obtain the following estimate for the integral

$$S_j^{(2)} = \frac{h_j}{12}\left[f(x_j) + 4f\left(x_j + \frac{h_j}{4}\right) + 2f\left(x_j + \frac{h_j}{2}\right) + 4f\left(x_j + \frac{3h_j}{4}\right) + f(x_j + h_j)\right] \quad (4.32)$$

The idea is to compare the two estimates, $S_j^{(1)}$ and $S_j^{(2)}$ to obtain as estimate for the accuracy of $S_j^{(2)}$. If the accuracy of $S_j^{(2)}$ is acceptable then we move to the next interval, otherwise we further subdivide the current interval until the accuracy requirement is met.

Let I_j denote the exact integral over the interval $[x_j, x_{j+1}]$ and using the error analysis we know that

$$I_j - S_j^{(1)} = ch_j^5 f^{(iv)}\left(x_j + \frac{h_j}{2}\right) + \dots \quad (4.33)$$

and

$$I_j - S_j^{(2)} = c\left(\frac{h_j}{2}\right)^5 \left[f^{(iv)}\left(x_j + \frac{h_j}{4}\right) + f^{(iv)}\left(x_j + \frac{3h_j}{4}\right)\right] + \dots \quad (4.34)$$

Expanding each of the terms in square brackets in a Taylor series about the point $(x_j + h_j/2)$

$$f^{(iv)}\left(x_j + \frac{h_j}{4}\right) = f^{(iv)}\left(x_j + \frac{h_j}{2}\right) - \frac{h_j}{4} f^{(v)}\left(x_j + \frac{h_j}{2}\right) + \dots \quad (4.35)$$

$$f^{(iv)}\left(x_j + \frac{3h_j}{4}\right) = f^{(iv)}\left(x_j + \frac{h_j}{2}\right) + \frac{h_j}{4} f^{(v)}\left(x_j + \frac{h_j}{2}\right) + \dots \quad (4.36)$$

allows us to rewrite equation (4.34) as

$$I_j - S_j^{(2)} = \frac{1}{16} ch_j^5 \left[f^{(iv)}\left(x_j + \frac{h_j}{2}\right)\right] + \dots \quad (4.37)$$

Subtracting (4.33) from (4.37) yields

$$S_j^{(2)} - S_j^{(1)} = \frac{15}{16} ch_j^5 f^{(iv)}\left(x_j + \frac{h_j}{2}\right) + \dots \quad (4.38)$$

Comparing equations (4.37) and (4.38) we conclude that the error in $S_j^{(2)}$ is approximately $1/15$ the difference between $S_j^{(2)}$ and $S_j^{(1)}$. Returning to our global error requirement in equation (4.30), if

$$\frac{1}{15}\left|S_j^{(2)} - S_j^{(1)}\right| \le \frac{h_j}{b-a}\epsilon \quad (4.39)$$

then $S_j^{(2)}$ is sufficiently accurate on the interval $[x_j, x_{j+1}]$ and we can move on to the next interval. Otherwise, the interval must be subdivided and the process repeated.

Adaptive quadrature is similar to Richardson extrapolation in that the form of the truncation error is used to obtain an estimate of the accuracy of the method without knowing the exact solution. Similar adaptive algorithms can be developed for other quadrature methods such as midpoint and trapezoidal rules.

4.4 Gauss Quadrature

Up to this point we have selected the values of the weights, w_j, in equation (4.2) to give the best approximation of the integral, I, assuming that the mesh points, x_j, are given. In Gauss quadrature, we extend upon this idea by selecting both the weights and the location of the mesh points to obtain the best accuracy. Here the measure of accuracy will be the highest degree polynomial that can be integrated exactly. By examining the form of the leading error term, it is easy to see that trapezoidal rule integrates a linear function exactly and Simpson's rule integrates a cubic exactly. We will see that Gauss quadrature integrates a polynomial of degree $2n+1$ exactly, using only $n+1$ points.

Let f be a polynomial of degree $2n+1$ and suppose that we interpolate f by an n^{th} order Lagrange polynomial, P, defined by the points x_0, x_1, x_2, ..., x_n. Using Lagrange interpolation yields

$$P(x) = \sum_{j=0}^{n} f(x_j) L_j^{(n)}(x) \tag{4.40}$$

If f had been a degree n polynomial then this representation would be exact. However, since f is a $2n+1$ degree polynomial, the difference, $f(x) - P(x)$, is also a polynomial of degree $2n+1$ which happens to vanish at the points x_0, x_1, x_2, ..., x_n. Thus we can write the difference $f(x) - P(x)$ as

$$f(x) - P(x) = F(x) \sum_{l=0}^{n} q_l x^l \tag{4.41}$$

where $F(x)$ is an $n+1$ order polynomial of the form

$$F(x) = (x - x_0)(x - x_1)(x - x_2) \cdots (x - x_n) \tag{4.42}$$

Integrating equation (4.41) leads to

$$\int f(x)dx = \int P(x)dx + \int F(x) \sum_{l=0}^{n} q_l x^l dx \tag{4.43}$$

By selecting the locations of the points x_0, x_1, x_2, $\ldots$, x_n we can demand that

$$\int F(x)x^l dx = 0 \qquad \text{for} \quad l = 0, 1, 2, \ldots, n \tag{4.44}$$

Choosing the abscissa in this manner eliminates the last term in equation (4.43) leading to

$$\int f(x)dx = \int P(x)dx = \sum_{j=0}^{n} f(x_j)w_j \tag{4.45}$$

where the weights are given by

$$w_j = \int L_n^{(n)}(x)dx \tag{4.46}$$

By construction, $F(x)$ is a polynomial of degree $n+1$ that is orthogonal to all polynomials of degree n or less, and the points x_0, x_1, x_2, $\ldots$, x_n are the zeros of this polynomial. If x varies between -1 and 1, then F is called a Legendre polynomial. Legendre polynomials are orthonormal meaning that

$$\int_{-1}^{1} F_n(x)F_m(x)dx = \delta_{nm} \tag{4.47}$$

where F_n is the Legendre polynomial of degree n.

The zeros and weights for the Legendre polynomials are documented in Gauss quadrature tables (see the CRC Standard Mathematical Tables). Note that it is always possible to transform the interval $a \le x \le b$ into $-1 \le \xi \le 1$ by the transformation

$$x = \frac{b+a}{2} + \frac{b-a}{2}\xi \tag{4.48}$$

To use Legendre-Gauss quadrature tables to evaluate the integral

$$\int_a^b f(x)dx \tag{4.49}$$

change the independent variable to ξ and determine the weights, w_j, and the points, ξ_j, from the tables for the given value of n (in the tables n denotes the number of points, not $n+1$). The integral is then approximated by

$$\frac{b-a}{2}\sum_{j=1}^{n} f\left(\frac{b+a}{2} + \frac{b-a}{2}\xi_j\right) w_j \tag{4.50}$$

Note that the values of the weights are symmetric about $\xi = 0$.

Although Legendre-Gauss quadrature is the most versatile of the Gauss quadrature methods, there are also special purpose formulae for some common integral forms:

$$\int_0^\infty e^{-x} f(x) dx \qquad \text{use Laguerre-Gauss quadrature} \tag{4.51}$$

$$\int_0^\infty e^{-x^2} f(x) dx \qquad \text{use Hermite-Gauss quadrature} \tag{4.52}$$

$$\int_0^\infty \frac{f(x)}{\sqrt{1-x^2}} dx \qquad \text{use Chebyshev-Gauss quadrature} \tag{4.53}$$

and values for the abscissa and weights for these quadrature formulae also appear in standard Gauss quadrature tables.

Chapter 5

Ordinary Differential Equations

Many physical processes in science and engineering can be mathematically expressed in terms of Ordinary Differential Equations (ODE). You may recall that a high-order ODE can be converted to a system of first ODE's. For example, the equation

$$\frac{d^2u}{dt^2} + \frac{du}{dt} = f(u,t) \tag{5.1}$$

can be rewritten as

$$w - \frac{du}{dt} = 0 \tag{5.2}$$

$$\frac{dw}{dt} + w = f(u,t) \tag{5.3}$$

which is a system of two coupled, first-order ODE's. The extension to higher-order ODE's is obvious. As we will see in the next chapter, the numerical solution of Partial Differential Equations (PDE) can also be expressed as the solution of a coupled system of first-order ODE's. Thus, the emphasis here will be on the solution of first-order ODE's and, as will be shown below, the extension to systems of first-order equation will be straight forward.

ODE can be divided into two classes depending on the form of the boundary conditions. If all the boundary data are prescribed at one value of the independent variable (say $t = 0$) then the ODE is called an initial value problem.

$$u'' = f(t,y) \qquad u(0) = u_0 \quad u'(0) = u'_0 \tag{5.4}$$

If, on the other hand, data are prescribed at more than one value of t then the problem is a boundary value problem. If should be obvious that boundary value problems require at least a second-order differential equation, such as

$$u'' = f(t,u,u') \qquad u(0) = u_0 \quad u(L) = u_L \tag{5.5}$$

where f is an arbitrary function. Of the two types of ODE problems, initial value problems are, in some sense, more fundamental since algorithms for boundary value problems are often built using algorithms for initial value problems. Thus, our discussion begins with the first-order initial value problem.

5.1 Initial Value Problems

Consider the differential equation

$$\frac{du}{dt} = u' = f(t,u) \qquad u(0) = u_0 \tag{5.6}$$

The objective of our numerical method is to obtain the solution at time $t_{n+1} = t_n + \Delta t$ given the solution for $0 \leq t \leq t_n$. In the following sections we will introduce and analyze a variety of numerical methods for solving this equation. As always, we will pay particular attention to the accuracy of the methods. In addition, we will see that some numerical methods for the solution of ODE can become unstable under certain circumstances.

5.1.1 Taylor Series Methods

Expanding the solution to equation (5.6) at t_{n+1} about the solution at t_n yields

$$u_{n+1} = u_n + hu'_n + \frac{h^2}{2}u''_n + \frac{h^3}{6}u'''_n + \dots \tag{5.7}$$

where $h = \Delta t$. We can use the differential equation to express u'_n in this equation in terms of $f(t_n, u_n)$.

$$u'_n = f(t_n, u_n) \tag{5.8}$$

The chain rule can be used to express the higher derivatives in equation (5.7) in terms of the function f. For example,

$$u'' = \frac{du}{dt} = \frac{df}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial u}\frac{du}{dt} = f_{,t} + ff_{,u} \tag{5.9}$$

Similarly for the third derivative

$$u''' = \frac{\partial}{\partial t}\left[f_{,t} + ff_{,u}\right] + \frac{\partial}{\partial u}\left[f_{,t} + ff_{,u}\right]f = f_{,tt} + 2ff_{ut} + f_{,t}f_{,u} + ff_{,u}^2 + f^2 f_{,uu} \tag{5.10}$$

Clearly, the number of terms in the derivative expressions increases rapidly making the method generally impractical for higher than second order.

Truncating after the first two terms, we obtain the explicit Euler (or simply Euler) scheme

$$u_{n+1} = u_n + hf(t_n, u_n) \tag{5.11}$$

To compute the numerical solution using this method, one starts with the initial condition, u_0, and simply marches using this formula to obtain u_1, u_2, etc. From the Taylor series expansion we see that the Euler scheme is second-order accurate over one time step. However, like the integration schemes already discussed, the error in advancing from the initial condition to the final time, T, is only first order. Thus, this scheme is often not accurate enough for practical engineering applications. However, due to its simplicity we will analyze this method extensively, comparing its characteristics to more complicated schemes.

To obtain higher accuracy, we must supply more information about the function f. In the Taylor series method this is accomplished by including derivatives of f. Another way of obtaining higher accuracy is to use intermediate values of f, between t_n and t_{n+1} (not including t_{n+1}). This leads to the family of *Runge–Kutta* schemes. Higher accuracy can also be obtained by using information from previous time steps, say t_{n-1}, t_{n-2}, etc. This leads to the family of *Multi-Step* methods.

All the methods introduced so far are *explicit* in that they do not involve f evaluated at t_{n+1}. Methods which do include information at t_{n+1} are called *implicit*. Since f may be a nonlinear function of u, implicit methods often require the solution of non-linear equations at each time step. Although this makes implicit methods more computationally expensive per time step, implicit methods can offer the advantage of numerical stability.

5.1.2 Numerical Stability and the Model Equation

Even though a numerical scheme may yield accurate solutions to an ODE such as

$$u' = f(u, t) \tag{5.12}$$

it is possible that the numerical solution can become unbounded even though the exact solution is bounded for all time. In stability analysis we wish to determine the conditions, in terms of the parameters of the numerical method (primarily the time step, h), for which the numerical solution is bounded. The stability characteristics of numerical methods for ODE can be divided into the following three categories:

1. **Stable:** the numerical solution remains bounded for any choice of numerical parameters.

2. **Unstable:** the numerical solution becomes unbounded for any choice of parameters.

3. **Conditionally stable:** there exist certain parameters for which the numerical solution is bounded.

Although equation (5.12) may be nonlinear depending on the form of $f(u,t)$, it is necessary, but not sufficient, for the numerical solution to (5.12) to be linearly stable. To examine the linear stability of (5.12) we expand $f(u,t)$ in a two-dimensional Taylor series:

$$\begin{gathered} f(u,t) = f(u_0,t_0) + (t-t_0)\frac{\partial f}{\partial t}(u_0,t_0) + (u-u_0)\frac{\partial f}{\partial u}(u_0,t_0) + \\ \frac{1}{2}\left[(t-t_0)^2\frac{\partial^2 f}{\partial t^2} + 2(t-t_0)(y-y_0)\frac{\partial^2 f}{\partial t \partial u} + (u-u_0)^2\frac{\partial^2 f}{\partial u^2}\right]_{(u_0,t_0)} + \dots \end{gathered} \tag{5.13}$$

which can be formally written as

$$u' = \lambda u + \alpha_1 + \alpha_2 t + \dots \tag{5.14}$$

where λ, α_1, and α_2 are constants. In many instances, the inhomogeneous terms in (5.14) do not affect the stability of the solution so that we can consider the model problem

$$u' = \lambda u \tag{5.15}$$

rather than the full problem (5.12). This greatly simplifies the stability analysis of numerical methods. However, there are situations where the stability characteristics of the nonlinear problem are not well predicted by the linear analysis. In these cases, numerical experimentation is necessary to determine the actual stability characteristics.

With this caveat, we proceed to consider the stability of various numerical methods applied to the model equation (5.15). For generality, we allow λ to be complex

$$\lambda = \lambda_r + i\lambda_i = \xi + i\omega \tag{5.16}$$

Doing so allows us to readily extend our analysis to systems of ODE's and PDE's. As an example, consider the second-order ODE

$$u'' + \omega^2 u = 0 \tag{5.17}$$

The exact solution is sinusoidal

$$u = c_1 \cos \omega t + c_2 \sin \omega t \tag{5.18}$$

We can convert this second order equation into a system of two first order equations of the form

$$\left\{ \begin{array}{c} u_1 \\ u_2 \end{array} \right\}' = \left[\begin{array}{cc} 0 & 1 \\ -\omega^2 & 0 \end{array} \right] \left\{ \begin{array}{c} u_1 \\ u_2 \end{array} \right\} \tag{5.19}$$

The reader is encouraged to verify that the eigenvalues of the 2×2 matrix are $\lambda = \pm i\omega$. From the theory of linear algebra, we can diagonalize the matrix

$$A = \left[\begin{array}{cc} 0 & 1 \\ -\omega^2 & 0 \end{array} \right] \tag{5.20}$$

with the matrix of eigenvectors, S

$$A = S^{-1} \Lambda S \tag{5.21}$$

which results in the uncoupled set of equations

$$z' = \Lambda z \tag{5.22}$$

where

$$z = S \left\{ \begin{array}{c} u_1 \\ u_2 \end{array} \right\} \tag{5.23}$$

and Λ is a diagonal matrix with the eigenvalues of A on the diagonal. The component equations can be written as

$$z_1' = i\omega z_1, \qquad z_2' = i\omega z_2 \tag{5.24}$$

This example demonstrates that higher-order ODE or systems of ODE can be reduced to uncoupled ODE which have the form of the model problem (5.15) but with complex λ. In general, the imaginary part of λ results in oscillatory solutions of the form

$$e^{\pm i\omega t} \tag{5.25}$$

while the sign of the real part of λ determines whether the solution grows or decays. In our stability analysis of the model equation, we will only be concerned with cases for which $\lambda_r \leq 0$ meaning that the amplitude of the exact solution either is constant or decays.

5.1.3 Explicit Euler

We begin our stability analysis by applying the Euler method

$$u_{n+1} = u_n + hf(u_n, t_n) \tag{5.26}$$

to the model equation (5.15) which leads to

$$u_{n+1} = u_n + \lambda h u_n = (1 + \lambda h)u_n \tag{5.27}$$

Thus the solution at time step n can be written as

$$u_n = (1 + \lambda h)^n u_0 \tag{5.28}$$

For complex λ, this can be written as

$$u_n = (1 + \lambda_r h + i\lambda_i h)^n u_0 = \sigma^n u_0 \tag{5.29}$$

where σ is called the amplification factor. For the numerical solution to be stable (i.e. remain bounded) requires that

$$|\sigma| \leq 1 \tag{5.30}$$

Since we require that the exact solution decays ($\lambda_r \leq 0$), the region of stability for the exact solution in the ($\lambda_r h$ – $\lambda_i h$) plane is the left half plane, shown in figure 5.1

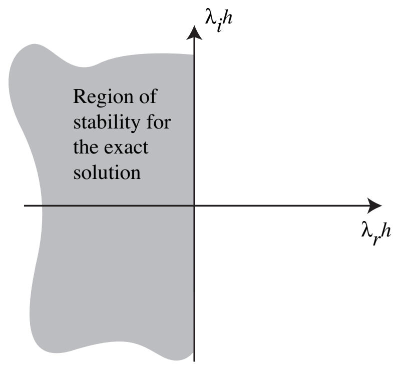

Figure 5.1: Stability diagram for the exact solution to the model equation.

However, when the Euler method is applied to the model problem, the numerical solution is only stable within the circle

$$(1 + \lambda_r h)^2 + (\lambda_i h)^2 = 1 \tag{5.31}$$

as shown in figure 5.2. For any value of λh in the left hand plane and outside this circle, the numerical solution will blow-up while the exact solution will decay. Thus, the Euler method is

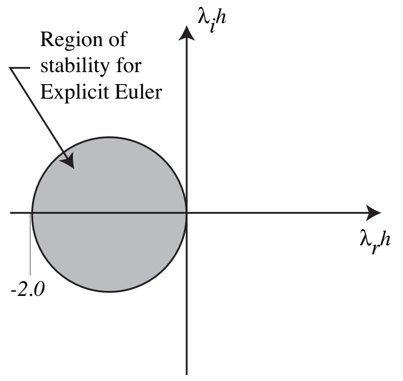

Figure 5.2: Stability diagram for explicit Euler scheme applied to the model equation.

conditionally stable. A stable numerical solution requires that we reduce h so that λh falls within the circle shown in figure 5.2. If λ is real, then the condition for numerical stability is

$$h \leq \frac{2}{|\lambda|} \tag{5.32}$$

Notice that the circle in figure 5.2 is only tangent to the imaginary axis. This means that the explicit Euler method is *always* unstable, regardless of the value of h, when λ is purely imaginary. As we will see in the following chapter, this renders the explicit Euler method unsuitable for problems which involve pure convection.

It is interesting to see what happens to the solution when the method becomes unstable. If λ is real and the method is unstable, then the following inequality must be true

$$|1 + \lambda h| > 1 \tag{5.33}$$

which, since $\lambda > 0$, means that $(1 + \lambda h)$ is negative with magnitude greater than 1. Since

$$u_n = (1 + \lambda h)^n u_0 \tag{5.34}$$

we see that the numerical solution oscillates with a change of sign at every time step. This type of behavior is indicative of numerical instability.

5.1.4 Implicit or Backward Euler

The next scheme we will analyze is the implicit Euler method given by the formula

$$u_{n+1} = u_n + hf(u_{n+1}, t_{n+1}) \tag{5.35}$$

This scheme is identical to explicit Euler, except that f is now evaluated at the new time step, $n + 1$. If f is non-linear, we must solve a non-linear algebraic equation at each time step to obtain the solution at $n + 1$. Thus, the computational cost per time step can be much greater than that of explicit Euler. However, implicit Euler has the advantage that it is unconditionally stable. Furthermore, the need for iteration at each time step can be avoided by linearization, as shown below.

Applying implicit Euler to the model equation (5.15) yields

$$u_{n+1} = u_n + \lambda h u_{n+1} \tag{5.36}$$

Solving for u_{n+1}, we obtain

$$u_{n+1} = (1 - \lambda h)^{-1} u_n = \sigma^n u_0 \tag{5.37}$$

Considering complex λ, we have

$$\sigma = \frac{1}{(1 - \lambda_r h) - i\lambda_i h} = Ae^{i\theta} \tag{5.38}$$

where

$$A = \frac{1}{\sqrt{(1 - \lambda_r h)^2 + (\lambda_i h)^2}} \qquad \theta = -\tan^{-1} \frac{\lambda_i h}{1 - \lambda_r h} \tag{5.39}$$

Since $\lambda_r \leq 0$, the modulus of σ is

$$|\sigma| = |A||e^{i\theta}| = A < 1 \tag{5.40}$$

Thus, we can conclude that implicit Euler is *unconditionally* stable. Unconditional stability is typical of implicit methods (although there are some implicit methods which are only conditionally stable). The price we pay is the higher computational cost per time step.

It is important to realize that numerical stability does *not* imply accuracy. A method can be stable but inaccurate. From the point of view of stability, our objective is to use the largest time step, h, possible to reach the final time $t = T$. Large time steps lead to less function evaluations and hence lower computational cost. However, it should come to no surprise that the larger the step size the less accurate the solution.

5.1.5 Numerical Accuracy: Magnitude and Phase Errors

The exact solution to the model problem

$$u' = \lambda u \tag{5.41}$$

is

$$u(t) = u_0 e^{\lambda t} = u_0 (e^{\lambda h})^n \tag{5.42}$$

while the numerical solution is of the form

$$u_n = u_0 \sigma^n \tag{5.43}$$

By comparing to the amplification factor σ to the exact solution, we can determine the order of accuracy of the method. For this purpose, we can expand the exponential term in the exact solution in a Taylor series

$$e^{\lambda h} = 1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \ldots \tag{5.44}$$

For example, the amplification factor for Euler is

$$\sigma = 1 + \lambda h \tag{5.45}$$

while the amplification factor for implicit Euler is

$$\sigma = \frac{1}{1 - \lambda h} = 1 + \lambda h + \lambda^2 h^2 + \lambda^3 h^3 + \ldots \tag{5.46}$$

Both methods only reproduce the first two terms of the Taylor series expansion of the exponential in the exact solution. Thus both methods are second-order accurate for one time step, but (similar to numerical integration), the methods are globally only first order. We will call a method α order accurate if the expansion of its amplification factor matches the terms up to and including the $\lambda^\alpha h^\alpha / \alpha!$ term in the exponential expansion.

In the case of oscillatory solutions, the order of accuracy by itself is not very informative. If we consider the model problem with λ pure imaginary

$$u' = i\omega u \qquad u(0) = 1 \tag{5.47}$$

then the exact solution is $e^{i\omega t}$ which is oscillatory. The frequency of the oscillation is ω and the amplitude is 1. The numerical solution with explicit Euler is

$$u_n = \sigma^n u_0 \tag{5.48}$$

where $\sigma = 1 + i\omega h$. Clearly $|\sigma| > 1$ which confirms that Euler is unstable for purely imaginary λ. Since σ is a complex number we can write it as

$$\sigma = |\sigma| e^{i\theta} \tag{5.49}$$

where

$$\theta = \tan^{-1} \omega h = \tan^{-1} \frac{Im(\sigma)}{Re(\sigma)} \tag{5.50}$$

The phase error is then defined as

$$PE = \omega h - \theta = \omega h - \tan^{-1} \omega h \tag{5.51}$$

Using the power series for $\tan^{-1}$

$$\tan^{-1} \omega h = \omega h - \frac{(\omega h)^3}{3} + \frac{(\omega h)^5}{5} - \frac{(\omega h)^7}{7} + \dots \tag{5.52}$$

we have

$$PE = \frac{(\omega h)^3}{3} + \dots \tag{5.53}$$

which corresponds to a phase lag. This phase error occurs at each time step, so the the phase error after n time steps is nPE.

5.1.6 Trapezoidal Method

Formally integrating equation (5.6) over one time step leads to

$$u_{n+1} = u_n + \int_{t_n}^{t_{n+1}} f(u, \tau) d\tau \tag{5.54}$$

If we approximate the integral using the trapezoidal rule then

$$u_{n+1} = u_n + \frac{h}{2} \left[f(u_{n+1}, t_{n+1}) + f(u_n, t_n) \right] \tag{5.55}$$

This is the trapezoidal method for the solution of ODE. When this method is applied to PDEs (see Chapter 6) it is commonly called the Crank–Nicholson method.

Applying the trapezoidal method to the model equation gives

$$u_{n+1} = u_n + \frac{h}{2}\left[\lambda u_{n+1} + \lambda u_n\right] \tag{5.56}$$

which we note is just the average of the explicit and implicit Euler schemes. Solving for u_{n+1}, we obtain

$$u_{n+1} = \frac{1+\frac{\lambda h}{2}}{1-\frac{\lambda h}{2}} u_n \tag{5.57}$$

and expanding the amplification factor leads to

$$\sigma = \frac{1+\frac{\lambda h}{2}}{1-\frac{\lambda h}{2}} = 1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{4} + \dots \tag{5.58}$$

which demonstrates that the method is globally second order accurate. This additional accuracy is obtained at virtually no extra cost over implicit Euler (Moral: don't use implicit Euler).

To examine the stability properties of the trapezoidal method, we compute the modulus of σ for complex λ.

$$\sigma = \frac{1+\frac{\lambda_r h}{2} + i\frac{\lambda_i h}{2}}{1-\frac{\lambda_r h}{2} - i\frac{\lambda_i h}{2}} \tag{5.59}$$

Both the numerator and denominator are complex and can be written, respectively, as $Ae^{i\theta}$ and $Be^{i\phi}$ where

$$A = \sqrt{\left(1+\frac{\lambda_r h}{2}\right)^2 + \left(\frac{\lambda_i h}{2}\right)^2} \tag{5.60}$$

$$B = \sqrt{\left(1-\frac{\lambda_r h}{2}\right)^2 + \left(\frac{\lambda_i h}{2}\right)^2} \tag{5.61}$$

Thus,

$$\sigma = \frac{A}{B}e^{i(\theta-\phi)} \tag{5.62}$$

or

$$|\sigma| = \frac{A}{B} \tag{5.63}$$

Since we are only interested in the case where $\lambda_r \leq 0$ and in this case $A < B$ it follows that

$$|\sigma| < 1 \tag{5.64}$$

making the trapezoidal method unconditionally stable. Note, however, that for real and negative λ,

$$\lim_{h\to\infty} \sigma = -1 \tag{5.65}$$

which shows that the numerical solution oscillates between -1 and 1 for large h. Although the solution will not blow-up, it is very inaccurate for large h.

Now lets consider the case of oscillatory solutions for which $\lambda = i\omega$. In this case $A = B$ and therefore

$$|\sigma| = 1 \tag{5.66}$$

indicating that there is *no* amplitude error for trapezoidal rule. Since

$$\sigma = e^{2i\theta} \qquad \theta = \tan^{-1}\left(\frac{\omega h}{2}\right) \tag{5.67}$$

the phase error is given by

$$PE = \omega h - 2\tan^{-1}\frac{\omega h}{2} = \omega h - 2\left[\frac{\omega h}{2} - \frac{(\omega h)^3}{24}\right] + \ldots = \frac{(\omega h)^3}{12} + \ldots \tag{5.68}$$

which is approximately a factor of four better than explicit Euler.

5.1.7 Linearization for Implicit Methods

The main difficultly with implicit methods is that they generally require the solution of a non-linear algebraic equation as each time step. This is most often accomplished using a Newton–Raphson type iteration. However, consistent with the accuracy of the numerical method, iteration can be avoided by the linearization technique. Consider the ODE

$$u' = f(u,t) \tag{5.69}$$

where f is a non-linear function of u. Using the trapezoidal method to solve this equation yields

$$u_{n+1} = u_n + \frac{h}{2}\left[f(u_{n+1},t_{n+1}) + f(u_n,t_n)\right] + O(h^3) \tag{5.70}$$

Solving for u_{n+1} would require solving a non-linear equation. However if we replace $f(u_{n+1}, t_{n+1})$ with the first two terms of the Taylor series expansion

$$f(u_{n+1}, t_{n+1}) = f(u_n, t_{n+1}) + (u_{n+1} - u_n)\frac{\partial f}{\partial u}\Big|_{u_n} + \frac{1}{2}(u_{n+1} - u_n)^2 \frac{\partial^2 f}{\partial u^2}\Big|_{u_n} + \dots \tag{5.71}$$

and note that

$$u_{n+1} - u_n = O(h) \tag{5.72}$$

then we are left with

$$u_{n+1} = u_n + \frac{h}{2}\left[f(u_n, t_{n+1}) + (u_{n+1} - u_n)\frac{\partial f}{\partial u}\Big|_{u_n} + f(u_n, t_n)\right] + O(h^3) \tag{5.73}$$

where the order of accuracy has not been affected. Rearranging and solving for u_{n+1} yields

$$u_{n+1} = u_n + \frac{h}{2}\left(\frac{f(u_n, t_{n+1}) + f(u_n, t_n)}{1 - \frac{h}{2}\frac{\partial f}{\partial u}\Big|_{u_n}}\right) \tag{5.74}$$

Remarks

1. The solution can proceed without iteration and with retention of global second order accuracy.
2. This procedure is identical to taking only one iteration of a Newton–Raphson algorithm.
3. Since our stability analysis was linear, we have not lost linear stability.

5.1.8 Runge–Kutta Methods

So far all the numerical methods we have examined only used function evaluations at n and $n + 1$. To increase the order of accuracy requires us to supply more information about the function f. One way of achieving this is to introduce intermediate points between t_n and t_{n+1} and include function evaluations at these points in our numerical method. This approach leads to the family of Runge–Kutta methods. Of course, the additional function evaluations result in higher cost per time step, however, it turns out that both the accuracy and stability properties are improved.

We begin with the general form of a second-order Runge–Kutta method for solving

$$u' = f(u, t) \tag{5.75}$$

The solution at time step t_{n+1} is given by

$$u_{n+1} = u_n + \gamma_1 k_1 + \gamma_2 k_2 \tag{5.76}$$

where the functions k_1 and k_2 are defined sequentially as

$$k_1 = hf(u_n, t_n) \tag{5.77a}$$

$$k_2 = hf(u_n + \beta k_1, t_n + \alpha h) \tag{5.77b}$$

and α, β, γ_1, and γ_2 are constants that need to be determined to ensure the highest possible order of accuracy for the method. The accuracy is established by considering the Taylor series expansion of $u(t_{n+1})$

$$u(t_{n+1}) = u_n + hu'_n + \frac{h^2}{2}u''_n + \dots \tag{5.78}$$

But, from the ODE we have

$$u'_n = f(u_n, t_n) \tag{5.79}$$

and using the chain rule we have already obtained in section 5.1.1

$$u''_n = f_{,t} + ff_{,u} \tag{5.80}$$

Thus,

$$u_{n+1} = u_n + hf(u_n, t_n) + \frac{h^2}{2}\left[f_{,t}(u_n, t_n) + f(u_n, t_n)f_{,u}(u_n, t_n)\right] + O(h^3) \tag{5.81}$$

A two-dimensional Taylor series expansion of k_2 in equation (5.77b) leads to

$$k_2 = h[f(u_n, t_n) + \beta k_1 f_{,u}(u_n, t_n) + \alpha h f_{,t}(u_n, t_n) + \dots] \tag{5.82}$$

Using this equations and (5.77a) in (5.76) yields

$$u_{n+1} = u_n + (\gamma_1 + \gamma_2)hf(u_n, t_n) + \gamma_2 \beta h^2 f(u_n, t_n) f_{,u}(u_n, t_n) + \gamma_2 \alpha h^2 f_{,t}(u_n, t_n) + \dots \tag{5.83}$$

Comparing (5.81) and (5.83) and matching coefficients of similar terms leads to

$$\gamma_1 + \gamma_2 = 1 \tag{5.84a}$$

$$\gamma_2\alpha = \frac{1}{2} \tag{5.84b}$$

$$\gamma_2\beta = \frac{1}{2} \tag{5.84c}$$

These are three non-linear equations for four unknowns. Using α as a free-parameter leads to

$$\gamma_2 = \frac{1}{2\alpha} \qquad \beta = \alpha \qquad \gamma_1 = 1 - \frac{1}{2\alpha} \tag{5.85}$$

This yields a one-parameter family of second order Runge–Kutta formulas:

$$k_1 = hf(u_n, t_n) \tag{5.86a}$$

$$k_2 = hf(u_n + \alpha k_1, t_n + \alpha h) \tag{5.86b}$$

$$u_{n+1} = u_n + \left(1 - \frac{1}{2\alpha}\right) k_1 + \frac{1}{2\alpha} k_2 \tag{5.86c}$$

The most often used second order Runge–Kutta formula is when $\alpha = 1/2$.

Runge–Kutta formulae are often presented in a different, fully equivalent, form called the predictor-corrector format. The second order Runge–Kutta formula with $\alpha = 1/2$ in predictor-corrector format is

$$u^*_{n+1/2} = u_n + \frac{h}{2} f(u_n, t_n) \tag{5.87a}$$

$$u_{n+1} = u_n + hf(u^*_{n+1/2}, t_{n+1/2}) \tag{5.87b}$$

We now consider the stability characteristics of this method by applying equations (5.86) to the model equation $u' = \lambda u$ with the result

$$k_1 = \lambda h u_n \tag{5.88a}$$

$$k_2 = h[\lambda u_n + \alpha\lambda^2 h u_n] = \lambda h[1 + \alpha h\lambda]u_n \tag{5.88b}$$

$$\begin{aligned} u_{n+1} &= u_n + \left(1 - \frac{1}{2\alpha}\right)\lambda h u_n + \frac{1}{2\alpha}\lambda h(1 + \alpha\lambda h)u_n \\ &= u_n\left(1 + \lambda h + \frac{\lambda^2 h^2}{2}\right) \end{aligned} \tag{5.88c}$$

From which it is clear that the method is second order accurate. The amplification factor is

$$\sigma = \left(1 + \lambda h + \frac{\lambda^2 h^2}{2}\right) \tag{5.89}$$

and for numerical stability we must have $|\sigma| \leq 1$. One way to satisfy this restriction is to set

$$\left(1 + \lambda h + \frac{\lambda^2 h^2}{2}\right) = e^{i\theta} \tag{5.90}$$

and find the complex roots, λh, of this polynomial for different values of θ. The resulting stability region is shown in figure 5.3. On the real axis the boundary is the same as that of explicit Euler; however, there is an improvement for complex λ. Similar to explicit Euler, second order Runge–Kutta is also unstable for pure imaginary $\lambda = i\omega$. In this case

$$|\sigma| = \sqrt{1 + \frac{\omega^4 h^4}{4}} > 1 \tag{5.91}$$

The phase error is given by

$$PE = \omega h - \tan^{-1}\left(\frac{\omega h}{1 - \frac{\omega^2 h^2}{2}}\right) \tag{5.92}$$

Expanding the last term leads to

$$PE = -\frac{\omega^3 h^3}{6} + \ldots \tag{5.93}$$

which is only a factor of two better than explicit Euler.

By far the most popular Runge–Kutta scheme is the fourth order formula

$$u_{n+1} = u_n + \frac{k_1}{6} + \frac{k_2 + k_3}{3} + \frac{k_4}{6} \tag{5.94a}$$

where

$$k_1 = hf(u_n, t_n) \tag{5.94b}$$

$$k_2 = hf(u_n + \frac{k_1}{2}, t_n + \frac{h}{2}) \tag{5.94c}$$

$$k_3 = hf(u_n + \frac{k_2}{2}, t_n + \frac{h}{2}) \tag{5.94d}$$

$$k_4 = hf(u_n + k_3, t_n + h) \tag{5.94e}$$

Note that to obtain fourth-order accuracy, four function evaluations are required at each time-step. If the method is applied to the model equation then

$$u_{n+1} = \left(1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24}\right) u_n \tag{5.95}$$

which confirms that the method is fourth order accurate. Again the stability diagram can be found

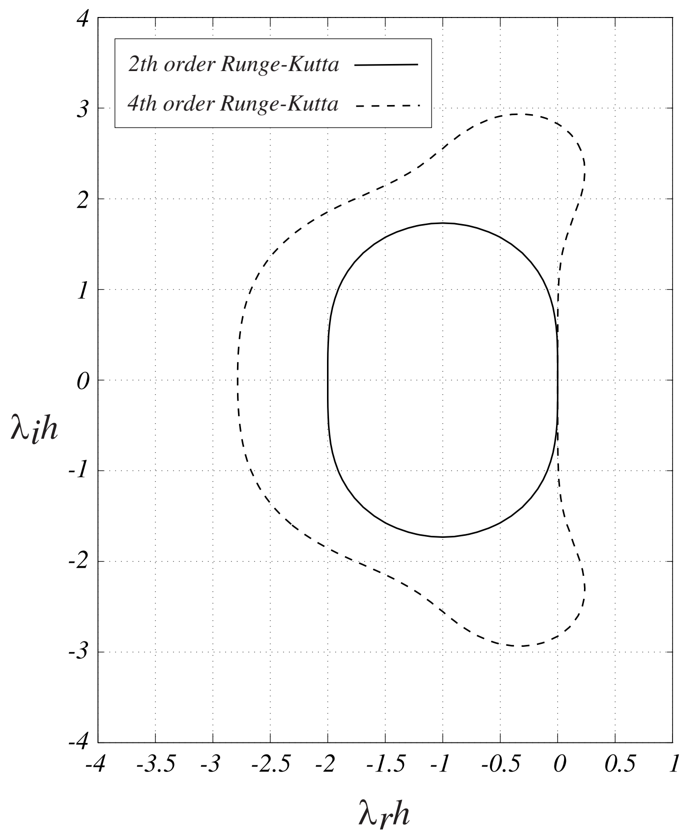

Figure 5.3: Stability diagram for second and fourth order Runge–Kutta schemes

by finding the zeros of the following polynomial which complex coefficients

$$1 + \lambda h + \frac{\lambda^2 h^2}{2} + \frac{\lambda^3 h^3}{6} + \frac{\lambda^4 h^4}{24} - e^{i\theta} = 0 \tag{5.96}$$

The stability region for this scheme is also shown in figure 5.3. There is a significant increase in the stable region compared to the second order Runge–Kutta and there is now a region of stability on the imaginary axis so that this method is viable for oscillatory solutions.

5.1.9 Multi-Step Methods

Runge–Kutta methods achieved higher order accuracy by using intermediate function evaluations. Increased accuracy can also be obtained if we use the solution and/or f from time steps prior to t_n. Methods which use prior information are called multi-step schemes. The price for the increased accuracy is that additional computer memory is required to store the information from prior time steps. Multi-step methods are *not* self-starting. Usually a self starting method is used to initiate the calculations for the first few time-steps.

Leapfrog method

A classic example of a multi-step method is the Leapfrog method

$$u_{n+1} = u_{n-1} + 2hf(u_n, t_n) \tag{5.97}$$

which is simply derived by applying the second order central difference formula to u' in equation (5.6). Applying Leapfrog to the model equation yields

$$u_{n+1} - u_{n-1} - 2\lambda h u_n \tag{5.98}$$

which is a difference equation for u_n. To solve this equation, we assume the standard solution of the form

$$u_n = \sigma^n \tag{5.99}$$

Substituting this into the difference equation leads to

$$\sigma^{n+1} - \sigma^{n-1} = 2h\lambda\sigma^n \tag{5.100}$$

which can be written as a quadratic equation for σ

$$\sigma^2 - 2h\lambda\sigma - 1 = 0 \tag{5.101}$$

The solution is

$$\sigma_{1,2} = \lambda h \pm \sqrt{\lambda^2 h^2 + 1} \tag{5.102}$$

Multiple roots are characteristic of multi-step methods. One root will approximate the exponential $e^{\lambda h}$ the other(s) will be spurious. Expanding the roots in (5.102) yields

$$\sigma_1 = \lambda h + \sqrt{\lambda^2 h^2 + 1} = 1 + \lambda h + \frac{\lambda^2 h^2}{2} - \frac{\lambda^4 h^4}{8} + \ldots \tag{5.103a}$$

$$\sigma_2 = \lambda h - \sqrt{\lambda^2 h^2 + 1} = -1 + \lambda h - \frac{\lambda^2 h^2}{2} + \frac{\lambda^4 h^4}{8} + \ldots \tag{5.103b}$$

In this case the first root approximates the exponential to second order accuracy while the second root is spurious and is often the source of numerical difficulties. Note that even when $h = 0$ the spurious root does not go to zero. Furthermore, for λ real and negative, the spurious root has magnitude greater than 1 which leads to instability for all h. In fact, Leapfrog is only stable for pure imaginary λ with $|\omega h| \leq 1$ as will be shown below.

Since the difference equation for u_n is linear, its general solution can be written as a linear combination of the roots

$$u_n = c_1\sigma_1^n + c_2\sigma_2^n \tag{5.104}$$

Thus, the solution is composed of contributions from both the physical and spurious roots. The constants c_1 and c_2 depend on the starting conditions

$$u_0 = c_1 + c_2 \qquad u_1 = c_1\sigma_1 + c_2\sigma_2 \tag{5.105}$$

Solving for c_1 and c_2 yields

$$c_1 = \frac{u_1 - u_0\sigma_2}{\sigma_1 - \sigma_2}, \qquad c_2 = \frac{u_0\sigma_1 - u_1}{\sigma_1 - \sigma_2} \tag{5.106}$$

Thus for the model problem, it is possible to suppress the spurious root if we set $u_1 = \sigma_1 u_0$. In general, the particular starting scheme used will determine the level of contribution of the spurious root, thus, care must be taken in selecting the starting scheme. However, even if the spurious root is suppressed initially, round-off errors due to finite precision math on a computer can re-start it.

If we apply Leapfrog to the case of pure imaginary $\lambda = i\omega$ then

$$\sigma_{1,2} = i\omega h \pm \sqrt{1 - \omega^2h^2} \tag{5.107}$$

If $|\omega h| < 1$ then $|\sigma_{1,2}| < 1$ and Leapfrog has no amplitude error. If $|\omega h| > 1$ then the method is unstable since

$$|\sigma_{1,2}| = \left|\omega \pm \sqrt{\omega^2h^2 - 1}\right| > 1 \tag{5.108}$$

The main attraction for Leapfrog is that within its (very restrictive) stability constraints, there is no amplitude error. However, an examination of the phase error shows that

$$PE = -\frac{\omega^3h^3}{6} + \ldots \tag{5.109}$$

which is approximately the same as the second order Runge–Kutta scheme.

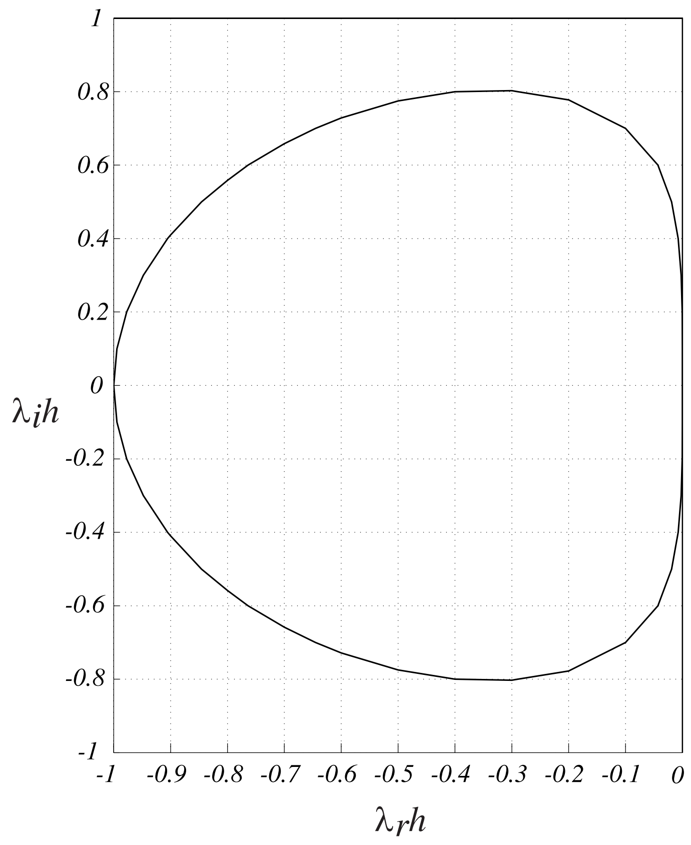

Figure 5.4: Stability diagram for second order Adams-Bashforth scheme

Adams–Bashforth Method

As another example of a multi-step method, we present the second order accurate Adams–Bashforth method. This method can be readily derived using a Taylor series expansion of u_{n+1} about u_n

$$u_{n+1} = u_n + hu'_n + \frac{h^2}{2}u''_n + \frac{h^3}{6}u'''_n + \dots \tag{5.110}$$

Substituting

$$u'_n = f(u_n, t_n) \tag{5.111}$$

and using the first order approximation for u''_n

$$u''_n = \frac{f(u_n, t_n) - f(u_{n-1}, t_{n-1})}{h} + O(h) \tag{5.112}$$

yields

$$u_{n+1} = u_n + \frac{3h}{2}f(u_n, t_n) - \frac{h}{2}f(u_{n-1}, t_{n-1}) + O(h^3) \tag{5.113}$$

which is the globally second order accurate Adams–Bashforth method.

Applying this method to the model equation generates a second order difference equation. Assuming solutions of the form $u_n = \sigma^n$ leads to a quadratic equation with roots

$$\sigma_{1,2} = \frac{1}{2}\left[1 + \frac{3}{2}\lambda h \pm \sqrt{1 + \lambda h + \frac{9}{4}\lambda^2 h^2}\right] \tag{5.114}$$

Expanding in a power series we obtain

$$\sigma_1 = 1 + \lambda h + \frac{1}{2}\lambda^2 h^2 + O(h^3) \tag{5.115}$$

$$\sigma_2 = \frac{1}{2}\lambda h - \frac{1}{2}\lambda^2 h^2 + O(h^3) \tag{5.116}$$

The spurious root appears to be less dangerous then that of the Leapfrog method since it goes to zero as h goes to zero. The stability region for the Adams–Bashforth method is shown in figure 5.4. We see that it crosses the real axis at -1 which makes it more limiting than explicit Euler and second order Runge–Kutta. Similar to these methods, the stability curve is only tangent to the imaginary axis, making the method, strictly speaking, unstable for pure imaginary λ. However, in practice, it turns out that the instability is very mild and the Adams–Bashforth method can be used to obtain oscillatory solutions as long as the integration time is not long enough that the instability becomes apparent.

One- and Two-Step Linear Multi-Step Methods

So far we have examined two particular multi-step methods. A thorough analysis of two-step multi-step methods has been performed by Beam and Warming (1978,1979) who wrote the most general two-step linear multi-step methods that is at least first order accurate as

$$(1+\xi)u_{n+1} = (1+2\xi)u_n - \xi u_{n-1} + h\left[\theta u'_{n+1} + (1-\theta+\psi)u'_n - \psi u'_{n-1}\right] \tag{5.117}$$

Clearly the methods are explicit if $\theta = 0$ and implicit otherwise. All of the schemes we have discussed, except Runge–Kutta methods, are included within this family of multi-step methods. For completeness, a list of the methods contained in (5.117) is given in table 5.1 using the names given by Beam and Warming.

θ	ξ	ψ	Method	Order
0	0	0	Euler	1
1	0	0	Implicit Euler	1
1/2	0	0	Trapezoidal	2
1	1/2	0	2nd Order Backward	2
3/4	0	-1/4	Adams type	2
1/3	-1/2	-1/3	Lees type	2
1/2	-1/2	-1/2	Two-step trapezoidal	2
5/9	-1/6	-2/9	A-contractive	2
0	-1/2	0	Leapfrog	2
0	0	1/2	2nd order Adams-Bashforth	2
0	-5/6	-1/3	Most accurate explicit	3
1/3	-1/6	0	3rd order implicit	3
5/12	0	1/12	3rd order Adams–Moulton	3
1/6	-1/2	1/6	Milne	4

Table 5.1: Linear one- and two-step methods

5.1.10 Systems of Ordinary Differential Equations

Systems of ordinary differential equations occur in a variety of engineering problems. We have already demonstrated that a high order ODE can be converted into a system of first order ODE. Systems of ODE's also arise naturally in mathematical models of many physics processes, such as the chemical reactions among several species. Furthermore, the numerical solution of partial differential equations naturally leads to a system of ODE that must be solved. Fortunately, we will show that the methods and analysis that we have developed for a single ODE can be readily extended to systems.

Lets begin with a generic system of ODE of the form

$$\mathbf{u}' = \mathbf{f}(\mathbf{u}, t) \tag{5.118}$$

where $\mathbf{u}$ is a vector with components u_i and $\mathbf{f}$ is a vector function with elements f_i. The application of a explicit numerical method to solve this equation is a straight forward extension of the techniques for single ODE. Applying explicit Euler to solve this equation leads to

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h\mathbf{f}(\mathbf{u}_n, t_n) \tag{5.119}$$

The right hand side can be calculated from data at the previous time step and the solution can be advanced forward.

In the numerical solution of systems of ODE, one component of the solution may oscillate very rapidly, while another component barely changes at all. This difference in the “response rates” of various components of the solution, is called *stiffness* and can lead to numerical difficulties. To explore the concept of stiffness, we introduce the following model problem for systems of ODE

$$\frac{d\mathbf{u}}{dt} = \mathbf{A}\mathbf{u} \tag{5.120}$$

where $\mathbf{A}$ is a $m \times m$ constant matrix. Similar to the model equation for single ODE’s, equation (5.120) is useful in analyzing numerical methods for systems. From linear algebra we know that the solution to this system will be bounded if all the eigenvalues of $\mathbf{A}$ have negative real parts (this is analogous to having negative λ_r for the single equation model problem, $u' = \lambda u$.)

Applying explicit Euler to equation (5.120) leads to

$$\mathbf{u}_{n+1} = \mathbf{u}_n + h\mathbf{A}\mathbf{u}_n = (\mathbf{I} + h\mathbf{A})\mathbf{u}_n \tag{5.121}$$

where $\mathbf{I}$ is the $m \times m$ identity matrix. Thus

$$\mathbf{u}_n = (\mathbf{I} + h\mathbf{A})^n \mathbf{u}_0 \tag{5.122}$$

and to have a bounded solution for large n the matrix $\mathbf{B}^n = (\mathbf{I} + h\mathbf{A})^n$ must approach zero. We know from linear algebra that for this to occur, the eigenvalues of $\mathbf{B}$ must have modulus less than one. The eigenvalues of $\mathbf{B}$ are given by

$$\beta_i = 1 + h\lambda_i \tag{5.123}$$

where λ_i are the eigenvalues of the matrix $\mathbf{A}$. For numerical stability we must have

$$|1 + \lambda_i h| \leq 1 \tag{5.124}$$

Therefore, the step size h must be chosen such that

$$|1 + \lambda_{max} h| \leq 1 \tag{5.125}$$

where the largest eigenvalue places the most stringent restriction on h. If the eigenvalues are all real and negative, then

$$h \leq \frac{2}{|\lambda_{max}|} \tag{5.126}$$

If the range of eigenvalues is large ($|\lambda|_{max}/|\lambda|_{min} >> 1$) and the solution is required over a large time space, then the system of equations is called a *stiff system*. Stiffness occurs when there are many degrees of freedom, each with widely different rates of response. Examples include: a system composed of two springs, one very stiff and the other very flexible; a mixture of chemical species with widely different reaction rates; and a boundary layer with widely different length scales.

Numerical difficulties arise with stiff systems when the solution is desired over a long period of time. For a stable numerical solution, the time step is limited by the component of the solution with the fastest response. However, if only the long term behavior, or steady-state solution, is desired, then the stability restriction due to the fast part of the solution will make the overall number of time steps required extremely high. To overcome this constraint, implicit methods are often used for stiff systems. Since they are generally unconditionally stable, the long term response can be determined by taking large time steps. Obviously, the fast components of the solution will not be resolved in this case. If the solution exhibits periods of fast response and periods of slow response, then the time step can be varied in the implicit method so that small steps are used for the fast portions and large time steps in the slow portion. Adaptive time step algorithms based on implicit methods, such as LSODE, have been developed to automatically select the time step based on the behavior of the solution. Note that explicit methods cannot take large time steps in the slowly varying portion of the solution since round-off errors will trigger numerical instability.

5.1.11 Linearization of Systems

When applying implicit methods to a single non-linear ODE we showed that linearization can be used to avoid iteration at each time step. A similar process can be performed for non-linear systems of equations. Consider the system

$$\mathbf{u}' = \mathbf{f}(u_1, u_2, \ldots, u_m) \tag{5.127}$$

Applying the trapezoidal method to this equation results in

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \frac{h}{2}\left[\mathbf{f}(\mathbf{u}_{n+1}) + \mathbf{f}(\mathbf{u}_n)\right] \tag{5.128}$$

The Taylor series expansion of $\mathbf{f}$ with elements f_i is

$$f_i(\mathbf{u}_{n+1}) = f_i(\mathbf{u}_n) + \sum_{k=1}^{m}\left[u_k^{(n+1)} - u_k^{(n)}\right]\left.\frac{\partial f_i}{\partial u_k}\right|_n + O(h^2) \qquad i = 1, 2, \ldots, m \tag{5.129}$$

which can be written in matrix form as

$$\mathbf{f}(\mathbf{u}_{n+1}) = \mathbf{f}(\mathbf{u}_n) + \mathbf{A}_n\left(\mathbf{u}_{n+1} - \mathbf{u}_n\right) + O(h^2) \tag{5.130}$$

where $\mathbf{A}_n$ is the Jacobian matrix defined as

$$\mathbf{A}_n = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \cdots & \frac{\partial f_1}{\partial u_m} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_m}{\partial u_1} & \frac{\partial f_m}{\partial u_2} & \cdots & \frac{\partial f_m}{\partial u_m} \end{bmatrix}_n \tag{5.131}$$

Consistent with the order of accuracy of the trapezoidal method, we can use the Taylor series expansion to form the following system of linear algebraic equations which must be solved at each time step

$$\left(\mathbf{I} - \frac{h}{2}\mathbf{A}_n\right)\mathbf{u}_{n+1} = \left(\mathbf{I} - \frac{h}{2}\mathbf{A}_n\right)\mathbf{u}_n + h\mathbf{f}(\mathbf{u}_n) \tag{5.132}$$

Note that the matrix $\mathbf{A}_n$ is a function of time and must be updated at each time step. Similar to the single equation case, this procedure is identical to taking one iteration of a Newton–Raphson nonlinear solver. As we have shown, only one such iteration is needed to retain global second-order accuracy; however, additional iterations will improve the "constant" in the leading error term.

5.2 Boundary Value Problems

As discussed in the introductory section of this Chapter, boundary value problems (BVP's) differ from initial value problems in that boundary condition data is specified at two different values of the independent variable. A second order boundary value problem is

$$u'' = f(x, u, u'), \qquad u(0) = u_0, \qquad u(L) = u_L \tag{5.133}$$

where f is an arbitrary function.

Numerical solutions to BVP's generally take one of two forms:

1. *Shooting Methods:* An iterative technique which uses the standard methods we have discussed for initial value problems.

2. *Direct Methods:* The derivatives are approximated by finite difference operators and the resulting system of equations are solved subject to the boundary conditions.

Since we have already developed technique for initial value problems, we begin by discussing shooting methods. First, let's reduce the second order equation (5.133) to a system of two first order equations

$$v = u' \tag{5.134}$$

$$u' = v \tag{5.135}$$

$$v' = f(x, u, v) \tag{5.136}$$

with the boundary conditions given in (5.133).

5.2.1 Shooting Methods

To solve this system of equations using one of the methods we have already discussed for initial value problems requires us to have starting conditions for both unknowns, u and v, at $x = 0$. Since we only have a condition for $u(0)$ we make a guess for $v(0)$ and integrate the equations to $x = L$. Unless we made a fortuitous guess the final value of $u(L)$ will not equal u_L. In this case, we update the value for $v(0)$ and repeat the process until $u(L) = u_L$. Of course the speed of convergence of this iterative process depends on the method that we use to update $v(0)$.

For linear problems the process is quite simple with only two iterations required. Consider the linear problem

$$u''(x) + A(x)u'(x) + B(x)u(x) = f(x) \tag{5.137a}$$

$$u(0) = u_0 \qquad u(L) = u_L \tag{5.137b}$$

If we denote two solutions of these equation as $u_1(x)$ and $u_2(x)$ formed by using two different initial guesses for $u'(0)$ then, since the equation is linear, the actual solution can be expressed as a linear combination

$$u(x) = c_1 u_1(x) + c_2 u_2(x) \tag{5.138}$$

Since $u_1(0) = u_2(0) = u(0)$ then

$$c_1 + c_2 = 1 \tag{5.139}$$

Satisfying the condition at $x = L$ leads to

$$c_1 u_1(L) + c_2 u_2(L) = u_L \tag{5.140}$$

Thus we have two equations in two unknowns and the solution is

$$c_1 = \frac{u_L - u_2(L)}{u_1 L - u_2 L} \qquad c_2 = \frac{u_1(L) - u_L}{u_1 L - u_2 L} \tag{5.141}$$

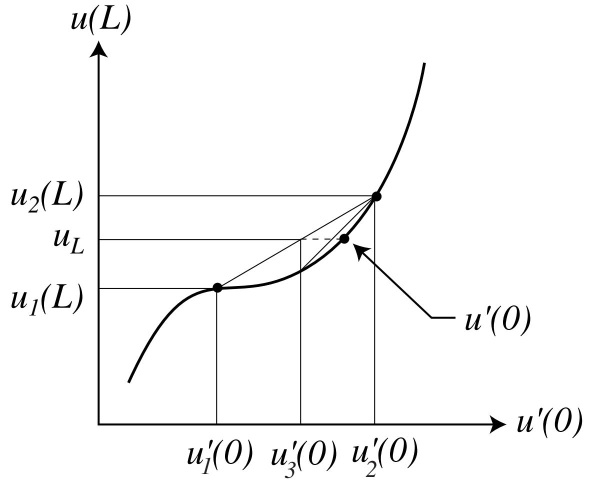

Figure 5.5: Secant method for solving non-linear boundary value problems using the shooting technique.

If, however, f is a non-linear function of u then we have to perform additional iterations to enforce the boundary condition at L to within a prescribed accuracy. Any number of iterative methods for solving non-linear equations can be used. Here we use the secant method. Consider the boundary value $u(L)$ as a nonlinear function of $u'(0)$. First we make two initial guesses for the starting derivative $u_1'(0)$ and $u_2'(0)$ and obtain the solutions $u_1(x)$ and $u_2(x)$ with the end values $u_1(L)$ and $u_2(L)$. The secant method for obtaining a new value for $u'(0)$ is illustrated in figure 5.5. A straight line is formed between the points $(u_1'(0), u_1(L))$ and $(u_2'(0), u_2(L))$. Then the intersection of this line with the line $u(L) = u_L$ is used as the next guess for $u'(0)$. This updated value is given by

$$u_3'(0) = u_1'(0) + \frac{u_2'(0) - u_1'(0)}{u_2(L) - u_1(L)}(u_L - u_1(L)) \tag{5.142}$$

This processes is repeated until the computed value of $u(L)$ is within a user prescribed tolerance. Convergence of the secant method may fail if $u(L)$ is a very sensitive function of $u'(0)$.

5.2.2 Direct Methods

In the direct method, the independent variable is discretized (*i.e.* a grid of node points in established) and one simply approximates the derivatives in the ODE with finite difference approximations of suitable accuracy. This results in a system of algebraic equations for the dependent variables at the node points. Linear ODE result in a system of linear algebraic equations while non-linear ODE lead to a system of non-linear algebraic equations. As an example, consider the second order accurate finite difference approximation of the linear equation (5.137):

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + A_j \frac{u_{j+1} - u_{j-1}}{2h} + B_j u_j = f_j \tag{5.143a}$$

$$u(j = 0) = u_0 \qquad u(j = N) = u_L \tag{5.143b}$$

Here we have assumed that the independent variable is discretized using a uniform grid, $x_j = x_{j-1} + h, j = 1, 2, \ldots, N - 1$, where $h = L/N$ and $x_0 = 0$, $x_N = L$. Equation (5.143a) can be rewritten as

$$\alpha_j u_{j+1} + \beta_j u_j + \gamma_j u_{j-1} = fj \tag{5.144}$$

where

$$\alpha_j = \left(\frac{1}{h^2} + \frac{A_j}{2h}\right) \qquad \beta_j = \left(B_j - \frac{2}{h^2}\right) \qquad \gamma_j = \left(\frac{1}{h^2} - \frac{A_j}{2h}\right) \tag{5.145}$$

$$j = 1, 2, 3, \ldots, N - 1 \tag{5.146}$$

This is a tri-diagonal system of linear equations. At the points next to the boundaries, $j = 1$ and $j = N - 1$ we take advantage of the known boundary values. For example,

$$\alpha_1 u_2 + \beta_1 u_1 = f_1 - \gamma_1 u_0 \tag{5.147}$$

and

$$\beta_{N-1} u_{N-1} + \gamma_{N-2} u_{N-2} = f_{N-1} - \alpha_{N-1} u_N \tag{5.148}$$

Thus we can rewrite the equations in the matrix form

$$\begin{bmatrix} \beta_1 & \alpha_1 & & & \\ \gamma_2 & \beta_2 & \alpha_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \gamma_{N-2} & \beta_{N-2} & \alpha_{N-2} \\ & & & \gamma_{N-1} & \beta_{N-1} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{Bmatrix} = \begin{Bmatrix} f_1 - \gamma_1 u_0 \\ f_2 \\ \vdots \\ f_{N-2} \\ f_{N-1} - \alpha_{N-1} u_N \end{Bmatrix} \tag{5.149}$$

and the solution can be obtained using the Thomas algorithm.

Higher-order finite difference approximations can also be used. The main difficulty in doing so arises near the boundaries where data are required from points outside the domain. The standard procedure is to use lower-order or one-sided difference formulas near the boundaries. Of course, increasing the order of the finite difference method will also increase the bandwidth of the matrix equation. For example, a fourth-order central difference approximation to (5.137) leads to a penta-diagonal matrix.

Non-Uniform Grids

It is often the case that the solution to an ODE varies rapidly in some regions of the domain and slowly in others. A typical example is that of a boundary layer where the solution gradients are large near the wall but approach zero in the freestream. If the regions of high gradients are known before hand (as is the case for the boundary layer) then the grid can be clustered where needed before the solution is obtained. If the regions of rapid variation are not known *a priori*, then an adaptive algorithm can be used that estimates the grid requirements based on the form of the truncation error (see section 5.1.10) and places additional grid points where needed.

When applying finite difference techniques, one can either use finite-difference formulae designed for non-uniform grids or a coordinate transformation. For example, common finite difference approximations on non-uniform grids for the first and second derivatives are given by

$$u'_j = \frac{u_{j+1} - u_{j-1}}{x_{j+1} - x_{j-1}} \tag{5.150}$$

$$u''_j = 2\left\{\frac{u_{j-1}}{h_j(h_j + h_{j+1})} - \frac{u_j}{h_j h_{j+1}} + \frac{u_{j+1}}{h_{j+1}(h_j + h_{j+1})}\right\} \tag{5.151}$$

where $h_j = x_j - x_{j-1}$. These formulae can be used in (5.137) for a non-uniform mesh leading again to a tri-diagonal system of equations. Generally speaking, the accuracy of finite-difference formulae for non-uniform meshes is less than that on a uniform mesh. This is due to reduced cancelations in the Taylor series expansions due to the lack of symmetry in the stencils. Care must be taken, when using finite-differences on non-uniform meshes, to ensure that the rate of change in h_j is not too large. Large changes in h_j can introduce numerical errors in the solution.

Another way of constructing difference formulas on a non-uniform mesh is to perform a transformation of the independent variable to another coordinate system which is chosen to account for local variations in the solution. Uniform spacing in the new (computational) coordinate system corresponds to non-uniform spacing in the original (physical) coordinate. An example of such a mapping is

$$\xi = \cos^{-1} x \tag{5.152}$$

which transforms the physical domain $0 \leq x \leq 1$ to $0 \leq \xi \leq \pi/2$. Uniform spacing in ξ, given by

$$\xi_j = \frac{j\pi}{2N} \qquad j = 0, 1, 2, \ldots, N \tag{5.153}$$

leads to a very fine mesh near $x = 1$ and a coarse mesh near $x = 0$. To transform the equation into the computational coordinate system, we write the mapping as

$$\xi = g(x) \tag{5.154}$$

and use the chain rule to transform the derivatives in the differential equation to the new coordinate system. Thus,

$$\frac{du}{dx} = \frac{d\xi}{dx}\frac{du}{d\xi} = g'\frac{du}{d\xi} \tag{5.155}$$

$$\frac{d^2u}{dx^2} = \frac{d}{dx}\left[g'\frac{du}{d\xi}\right] = g''\frac{du}{d\xi} + (g')^2\frac{d^2u}{d\xi^2} \tag{5.156}$$

These expressions are substituted into the differential equation and the derivatives with respect to ξ are replaced with finite difference approximations on a uniform mesh.

In general, the coordinate transformation method leads to more accurate solutions then the application of finite-difference operators for non-uniform grids. In addition, one can generally get away with more severe variations in the mesh spacing when using the coordinate transformation technique. The disadvantage of the coordinate transformation method is that the resulting differential equation can become rather complicated in the transformed coordinate, especially when high-order derivatives are present.

5.3 Differential Eigenvalue Problems

A problem related to boundary values problems is the differential eigenvalue problem. Here we are given a homogeneous ODE with homogeneous boundary conditions and we seek the natural, or fundamental, modes of the system. For example, consider the differential equation

$$\frac{d^2u}{dx^2} + k^2u = 0 \tag{5.157}$$

with boundary conditions

$$u(0) = 0 \qquad u(1) = 0 \tag{5.158}$$

The solution to this equation has the form

$$u(x) = a\,\sin(kx) + b\,\cos(kx) \tag{5.159}$$

Enforcing the first boundary condition leads to $b = 0$ while the second boundary condition requires that $a\sin(k) = 0$. We could require that $a = 0$ but this would lead to a trivial solution. Instead, we require that $k = j\pi$ where $j = 0, 1, 2, \ldots$ Here k is the eigenvalue and $u(x) = \sin(kx)$ is the eigenfunction for this differential eigenvalue problem. There are a countably infinite set of discrete eigenvalues and associated with each eigenvalue is an eigenfunction. Note that the larger the eigenvalue, the more oscillatory the eigenfunction. Together, each eigenvalue-eigenfunction pair is called an eigenmode.

To solve a differential eigenvalue problem numerically, we can replace the derivatives with finite difference approximations. Using the second order central difference for the second derivative in (5.157) leads to

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + k^2 u_i = 0 \tag{5.160}$$

$$y_0 = y_n = 0 \tag{5.161}$$

This can be written as the tri-diagonal system

$$\mathbf{B}\left[\frac{1}{h^2}; \frac{-2}{h^2} + k^2; \frac{1}{h^2}\right]\mathbf{u} = \mathbf{0} \tag{5.162}$$

where $\mathbf{B}[a; b; c]$ is a compact notation for a tri-diagonal matrix with constant diagonals.

For a nontrivial solution the determinant of the matrix must be zero. This is equivalent to

$$\det[k^2\mathbf{I} - \mathbf{A}] = 0 = \det[\lambda\mathbf{I} - \mathbf{A}] \tag{5.163}$$

where $\lambda = k^2$ and $\mathbf{A} = -\mathbf{B}[1; -2; 1]/h^2$. It can be shown that the eigenvalues of a $(N-1) \times (N-1)$ tri-diagonal matrix $\mathbf{B}[a, b, c]$ are given by the formula

$$\lambda_j = b + 2\sqrt{ac}\cos\alpha_j \tag{5.164}$$

$$\alpha_j = \frac{j\pi}{n} \qquad j = 1, 2, 3, \ldots, n-1 \tag{5.165}$$

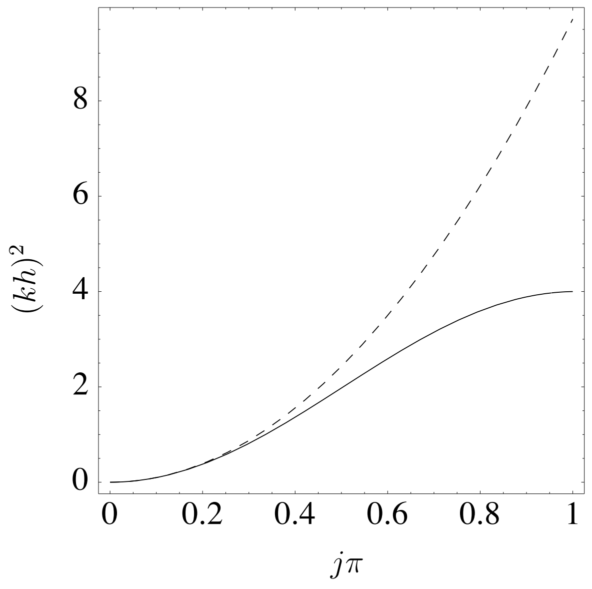

Figure 5.6: Comparison of numerical —— and analytical ---- eigenvalues for $n = 32$.

Thus, the eigenvalues of $\mathbf{A}$ are

$$\lambda_j = \frac{1}{h^2}\left(2 + 2\cos\frac{j\pi}{n}\right) \qquad j = 1, 2, 3, \ldots, n-1 \tag{5.166}$$

This expression is compared to the exact eigenvalues in figure 5.6 for $n = 32$. We see that the low-order numerical eigenvalues are in good agreement with the exact eigenvalues. As j gets larger, the numerical eigenvalues become increasingly poor approximations to the exact eigenvalues. This disagreement indicates that the eigenfunctions associated with these eigenvalues cannot be resolved on the given mesh.

Chapter 6

Partial Differential Equations

Partial differential equations (PDE's) can be classified as either equilibrium problems or propagation problems depending on the physical processes that they represent. Figure 6.1 shows schematics for each type of problem. Equilibrium problems are posed on a closed domain with fixed data supplied on the boundary. The problem consists of finding the steady state solution in the domain subject to the boundary conditions. Examples of equilibrium problems include: steady viscous flow, steady heat conduction, and equilibrium stress in a plate. Equilibrium problems are analogous to boundary value problems for ODE's.

PDE problems can also be posed as propagation problems that are transient, or unsteady, in nature. Propagation problems are analogous to initial value problems for ODE's. They are solved on an open domain starting from an initial condition and are subject to boundary conditions as shown in figure 6.1. Typical propagation problems include: unsteady heat convection, acoustic wave propagation, and vortex shedding from bluff bodies.

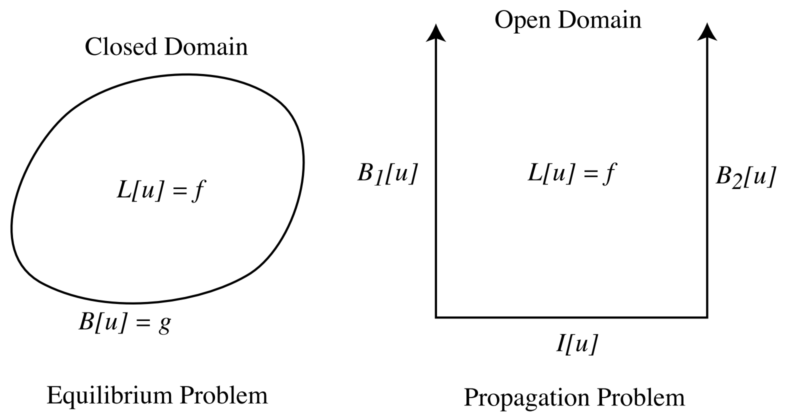

Figure 6.1: Physical classification of partial differential equations – equilibrium and propagation problems.

Another common means of classifying PDE's is according to the manner in which information is communicated throughout the domain. This leads to the familiar categories of hyperbolic, parabolic, and elliptic equations. A classical example of a hyperbolic PDE is the wave equation

$$\frac{\partial^2 u}{\partial t^2} - a^2 \frac{\partial^2 u}{\partial x^2} = 0 \tag{6.1}$$

This equation has two real characteristics

$$\frac{dt}{dx} = \pm\frac{1}{a} \tag{6.2}$$

Figure 6.2a shows a schematic of the characteristics for this hyperbolic problem. We see that

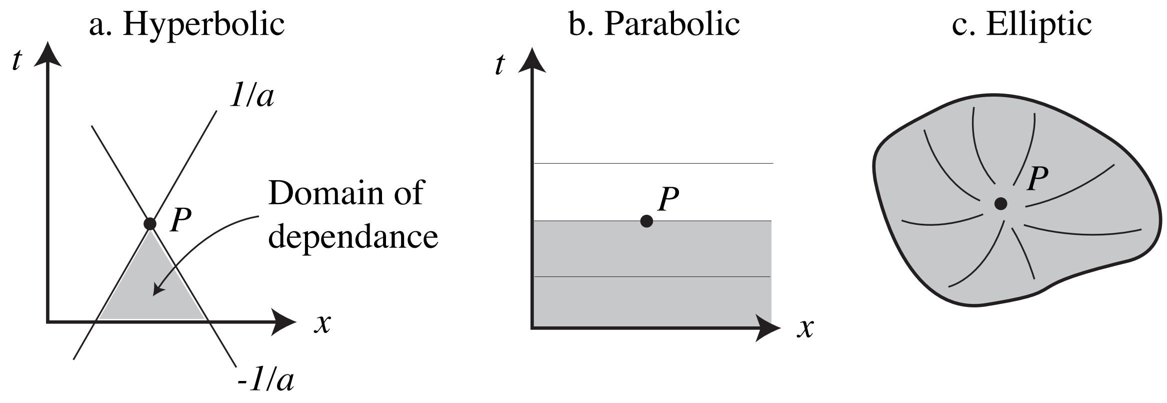

Figure 6.2: Classification of PDE: hyperbolic, parabolic, and elliptic.

the solution at the point P only influenced by the solution inside the shaded region – called the "domain of dependence." A schematic for a parabolic problem is given in figure 6.2b which shows that, in this case, the solution at P is influenced by all the data in x for previous time. Finally, figure 6.2c shows a typical elliptic problem, such as Laplace's equation

$$\nabla^2 u = 0 \tag{6.3}$$

For an elliptic problem, the solution at P is influenced by all the points in the domain.

6.1 Semi-Discretization and Matrix Stability Analysis

In previous chapters we have indicated that when partial differential equations are solved numerically, they lead to a system of ordinary differential equations. As an example, consider the one-dimensional heat equation

$$\frac{\partial u}{\partial t} = \nu\frac{\partial^2 u}{\partial x^2} \tag{6.4}$$

with boundary and initial conditions

$$u(0,t) = 0 \qquad u(L,t) = 0 \qquad u(x,0) = u_0(x) \tag{6.5}$$

If we approximate the spatial derivative in (6.4) using the second order central difference we obtain

$$\frac{\partial u_j}{\partial t} = \nu \frac{u_{j-1} - 2u_j + u_{j+1}}{\Delta x^2} \qquad j = 1, 2, \ldots, N-1 \tag{6.6}$$

where we have assumed that the domain is divided into $N+1$ uniformly spaced node points given by $x_j = jL/N$ for $j = 0, 1, 2, \ldots, N$. This is an example of semi-discretization where we have discretized in space but not in time. We can rewrite (6.6) as a $(N-1) \times (N-1)$ matrix equation

$$\frac{\partial}{\partial t} \left\{ \begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{array} \right\} = \frac{\nu}{\Delta x^2} \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \left\{ \begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{array} \right\} \tag{6.7}$$

This is clearly a system of ordinary differential equations that can be written in the form of our model ODE for systems

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{A}\mathbf{u} \tag{6.8}$$

where

$$\mathbf{A} = \frac{\nu}{\Delta x^2} \mathbf{B}[1, -2, 1] \tag{6.9}$$

and $\mathbf{B}[1, -2, 1]$ is a compact notation for a tri-diagonal matrix with constant diagonals.

This system of ordinary differential equations can be solved using any of the methods that we have discussed, such as Runge–Kutta formulae or multi-step methods. However, we must ensure that the method we use is stable and that stiffness does not render it unusable. Both stability and stiffness can be determined from the eigenvalue structure of the matrix $\mathbf{A}$. As shown in equation (5.164), the eigenvalues of the $(N-1) \times (N-1)$ tri-diagonal matrix $\mathbf{B}[a, b, c]$ are given by the formula

$$\lambda_j = b + 2\sqrt{ac} \cos \alpha_j \tag{6.10}$$

$$\alpha_j = \frac{j\pi}{N} \qquad j = 1, 2, 3, \ldots, N-1 \tag{6.11}$$

Thus, the eigenvalues of $\mathbf{A}$ are

$$\lambda_j = \frac{\nu}{\Delta x^2}\left(-2 + 2\cos\frac{\pi j}{N}\right) \qquad j = 1, 2, 3, \ldots, N-1 \tag{6.12}$$

Notice that all the eigenvalues are real and negative so that the solution decays. From our analysis of ODE methods, we can conclude that Leapfrog would be unstable if applied to (6.8). To determine stiffness we need the eigenvalues with the smallest and largest magnitudes. The eigenvalue with the smallest magnitude occurs for $j = 1$

$$\lambda_1 = \frac{\nu}{\Delta x^2}\left(-2 + 2\cos\frac{\pi}{N}\right) \tag{6.13}$$

We can simplify this expression by expanding the cosine term in a Taylor series for large N

$$\cos\frac{\pi}{N} = 1 - \frac{1}{2!}\left(\frac{\pi}{N}\right)^2 + \frac{1}{4!}\left(\frac{\pi}{N}\right)^4 + \ldots \tag{6.14}$$

If we retain only the first two terms then

$$\lambda_1 \approx -\frac{\pi^2\nu}{N^2\Delta x^2} \tag{6.15}$$

The eigenvalue with the largest magnitude occurs for $j = N-1$ and is given by

$$\lambda_{N-1} = -\frac{4\nu}{\Delta x^2} \tag{6.16}$$

Thus, the ratio of the largest to smallest magnitude eigenvalues is

$$\left|\frac{\lambda_{N-1}}{\lambda_1}\right| \approx \frac{4N^2}{\pi^2} \tag{6.17}$$

As N is increased the system clearly becomes stiff. Thus, we may wish to use an implicit scheme such as trapezoidal method as opposed to an explicit method like fourth-order Runge-Kutta.

Since all the eigenvalues are unique, we can use the standard methods of linear algebra to decouple the equations using the matrix of eigenvectors $\mathbf{S}$.

$$\frac{d\boldsymbol{\psi}}{dt} = \boldsymbol{\Lambda}\boldsymbol{\psi} \tag{6.18}$$

where $\boldsymbol{\Lambda} = \mathbf{S}^{-1}\mathbf{A}\mathbf{S}$ is a diagonal matrix with the eigenvalues of $\mathbf{A}$ on the diagonal and $\boldsymbol{\psi} = \mathbf{S}^{-1}\mathbf{u}$.

Since the equations are uncoupled in this form, the solution is simply

$$\psi_j(t) = \psi_j(0)e^{\lambda_j t} \qquad j = 1, 2, \ldots, N-1 \tag{6.19}$$

where $\boldsymbol{\psi}(0) = \mathbf{S}^{-1}\mathbf{u}_0$. Since all the eigenvalues, λ_j are real and negative, the solution decays as is expected for the heat equation.

Now we consider the model equation for pure convection, called the one-dimensional wave equation,

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{6.20}$$

$$u(0,t) = 0 \qquad u(L,t) = 0 \qquad u(x,0) = u_0(x) \tag{6.21}$$

Note, that the second boundary condition is not physically required, but we have to have some condition to close the numerical calculation. Setting $u(L,t) = 0$ is equivalent to stating that the integration time is short enough that the non-zero region of the initial condition does not hit the boundary.

We semi-discretize the wave equation using the second order central difference formula for the first derivative with the result

$$\frac{\partial u}{\partial t} + \frac{c}{2\Delta x}\left(u_{j+1} - u_{j-1}\right) = 0 \tag{6.22}$$

Rewriting in matrix notation leads to

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{c}{2\Delta x}\mathbf{B}[-1, 0, 1]\mathbf{u} \tag{6.23}$$

We can use equation (6.10) to determine the eigenvalues of this matrix

$$\lambda_j = -\frac{ic}{2\Delta x}\cos\frac{j\pi}{N} = i\omega_j \tag{6.24}$$

For this semi-discrete wave equation, the eigenvalues are purely imaginary which means that the solution does not decay, but instead oscillates like $e^{i\omega t}$ in time. Similar to the analysis for the heat equation above, we can use a similarity transformation to write the solutions in the form

$$\psi_j(t) = \psi_j(0)e^{i\omega_i t} \qquad j = 1, 2, \ldots, N-1 \tag{6.25}$$

The discretization of these two model equation provides support for the use of the model ordinary differential equation

$$u' = \lambda u \tag{6.26}$$

introduced in Chapter 5. The case with λ real and negative is a model for the heat equation (6.4) while the case with λ purely imaginary is a model for the wave equation 6.22. Thus, the application of numerical methods to solve the semi-discrete systems of ODE for these equations must address the same issues of stability, stiffness, and accuracy discussed in Chapter 5.

As an example, if we applied explicit Euler to equation (6.22) then, since the eigenvalues of the semi-discrete equation are pure imaginary, we expect numerical solutions that are *unconditionally unstable*. On the other hand, the application of explicit Euler to the semi-discrete heat equation (6.4) results in conditional stability. In this case, the maximum allowable time step is given by

$$|1 + \Delta t \lambda_i| \leq 1 \qquad i = 1, 2, 3, \ldots, N-1 \tag{6.27}$$

which leads to the requirement

$$\Delta t \leq \frac{\Delta x^2}{2\nu} \tag{6.28}$$

where we have used equation (6.16) for the maximum magnitude eigenvalue of the semi-discrete heat equation. Unfortunately this is a rather stringent constraint. It implies that if we decrease the spatial mesh size (to improve accuracy) by a factor of 2 then we have to decrease the time step by a factor of 4!

The analysis performed here, is called *matrix stability analysis*. It is a particularly powerful method since it includes the boundary conditions in the semi-discretization. Although many stability problems are due to the interior difference scheme, boundary conditions can adversely affect numerical stability.

In the following sections, we will introduce simpler methods for determining the stability properties of the interior difference scheme. However, the matrix stability analysis will remain our primary tool for determining the stability characteristics of the full semi-discrete problem including boundary conditions.

As a note of caution, the reader should recall that the results presented here are only valid when second order finite difference operators used for the spatial derivatives. We could have used first order or fourth order approximations — its up to you. However, doing so will lead to different matrix equations with different eigenvalue structures. For more complicated matrix equations, analytical solutions for the eigenvalues may not be readily available. In that case, algorithms for computing the eigenvalues of matrices are available in Matlab, Mathematica, and in the LAPACK library of FORTRAN subroutines.

6.2 von Neumann Analysis

If we are only concerned with the stability characteristics of a numerical method, without accounting for boundary conditions, then a simple method called von Neumann analysis is available. This technique assumes that the boundary conditions are periodic and is only valid for linear *constant coefficient* difference equations. Thus von Neumann's method is applied to the fully-discrete equation.

To demonstrate von Neumann's method, consider the fully discrete heat equation given by

$$u_j^{(n+1)} = u_j^{(n)} + \frac{\nu\Delta t}{\Delta x^2}\left(u_{j+1}^{(n)} - 2u_j^{(n)} + u_{j-1}^{(n)}\right) \tag{6.29}$$

The reader should recognize this as second order central in space and explicit Euler in time. The key to von Neumann's analysis is assuming that the solution to the difference equation is of the form

$$u_j^{(n)} = \sigma^n e^{ikx_j} \tag{6.30}$$

Recall that $e^{ikx_j} = \cos(kx_j) + i\sin(kx_j)$ which demonstrates the periodicity of the assumed solution. Substituting this solution into (6.29) results in

$$\sigma^{n+1}e^{ikx_j} = \sigma^n e^{ikx_j} + \frac{\nu\Delta t}{\Delta x^2}\sigma^n\left(e^{ikx_{j+1}} - 2e^{ikx_j} + e^{ikx_{j-1}}\right) \tag{6.31}$$

Noting that

$$x_{j+1} = x_j + \Delta x \qquad \text{and} \qquad x_{j-1} = x_j - \Delta x \tag{6.32}$$

and dividing through by $\sigma^n e^{ikx_j}$ we obtain

$$\sigma = 1 + \left(\frac{\nu\Delta t}{\Delta x^2}\right)[2\cos(k\Delta x) - 2] \tag{6.33}$$

The stability condition for explicit Euler is $|\sigma| \leq 1$ which means that

$$\left|1 + \left(\frac{\nu\Delta t}{\Delta x^2}\right)[2\cos(k\Delta x) - 2]\right| \leq 1 \tag{6.34}$$

This inequality can be written as

$$-1 \leq 1 + \left(\frac{\nu\Delta t}{\Delta x^2}\right)[2\cos(k\Delta x) - 2] \leq 1 \tag{6.35}$$

Since $[2\cos(k\Delta x) - 2] \leq 0$, the right-hand inequality is always satisfied. The left-hand inequality leads to the constraint

$$\Delta t \leq \frac{2\Delta x^2}{\nu[2 - 2\cos(k\Delta x)]} \tag{6.36}$$

The worst case occurs when $\cos(k\Delta x) = -1$ so that the maximum time step for stability is given by

$$\Delta t \leq \frac{\Delta x^2}{2\nu} \tag{6.37}$$

This is equivalent to equation (6.28) which was obtained using matrix stability analysis. The agreement is just a coincidence since the matrix analysis used Dirichlet boundary conditions while the von Neumann's method assumes periodicity.

To summarize, von Neumann's analysis determines the stability characteristics of a fully discrete numerical method applied to a linear PDE with constant coefficients and assuming periodicity.

6.3 Modified Wave Number Analysis

Modified wave number analysis is very similar to von Neumann analysis, but, in many ways is easier to perform and interpret. Basically, it allows us to use our knowledge about stability properties of numerical methods applied to ODE to problems involving the same numerical method applied to PDE.

As an example, consider the heat equation (6.4) and assume a solution of the form

$$u(x,t) = \psi(t)e^{ikx} \tag{6.38}$$

Substituting into (6.4) leads to

$$\frac{d\psi}{dt} = -\nu k^2\psi \tag{6.39}$$

where k is the wave number of the assumed solution. In practice, we must approximate the spatial derivative using a finite difference formula. If we use the second order central difference scheme then

$$\frac{\partial u_j}{\partial t} = \nu\frac{u_{j+1} - 2u_j + u_{j-1}}{\Delta x^2} \qquad j = 1, 2, 3, \ldots, N-1 \tag{6.40}$$

Let the solution to these semi-discrete equations be of the form

$$u_j = \psi(t)e^{ikx_j} \tag{6.41}$$

Substituting this form of the solution and dividing through by e^{ikx_j} leads to

$$\frac{\partial \psi}{\partial t} = -\frac{2\nu}{\Delta x^2}\left[1 - \cos(k\Delta x)\right]\psi = -\nu k'^2 \psi \tag{6.42}$$

where

$$k'^2 = \frac{2}{\Delta x^2}\left[1 - \cos(k\Delta x)\right] \tag{6.43}$$

By analogy to equation (6.39), k' is called the *modified wave number*. The modified wave number is a function of both the original PDE and the finite-difference scheme used. For a given equation, each finite-difference scheme has a distinct modified wave number associated with it.

Equation (6.42) is identical to the model equation $\psi' = \lambda\psi$ with $\lambda = -\nu k'^2$. Since we have studied the stability properties of various numerical methods applied to the model equation, we can readily use these results to obtain stability characteristics of time advancement schemes applied to PDE's. All we have to do is use $\lambda = -\nu k'^2$ in our ODE analysis.

For example, explicit Euler requires that

$$\Delta t \leq \frac{2}{|\lambda|_{max}} = \frac{2}{\left|\frac{2\nu}{\Delta x^2}[1 - \cos(k\Delta x)]\right|_{max}} \tag{6.44}$$

The worst case scenario occurs when $\cos(k\Delta x) = -1$. Thus

$$\Delta t \leq \frac{\Delta x^2}{2\nu} \tag{6.45}$$

which agrees with the results found using von Neumann analysis. If we decided to use fourth order Runge–Kutta then the stability limit would be

$$\Delta t \leq \frac{2.81\Delta x^2}{4\nu} \tag{6.46}$$

where the quantity 2.81 is read directly from the intersection of the stability contour with the negative real axis in figure 5.3. Since $-\nu k'^2$ is real and negative we also know that the Leapfrog method will lead to numerical instability.

Applying the modified wave number analysis to the wave equation (6.22) with second order central differencing in space leads to the following

$$\frac{\partial u}{\partial t} = -ik'cu \tag{6.47}$$

where

$$k' = \frac{\sin(k\Delta x)}{\Delta x} \tag{6.48}$$

is the modified wave number. Thus, the corresponding λ in the model equation $\lambda = -ik'c$ is pure imaginary. So without any further analysis, we know that explicit Euler and second order Runge–Kutta would be unstable for this problem. However, if Leapfrog is used the the maximum allowable time step is determined from

$$\Delta t = \frac{1}{k'c} = \frac{\Delta x}{c\,\sin(k\Delta x)} \tag{6.49}$$

and taking the worst case scenario, this leads to

$$\Delta t_{max} = \frac{\Delta x}{c} \tag{6.50}$$

or, if we form a nondimensional parameter,

$$\frac{c\Delta t}{\Delta x} \leq 1 \tag{6.51}$$

where $c\Delta t/\Delta x$ is the Courant, Friedrich, and Lewy number, CFL for short. The CFL number is a very important numerical parameter governing the stability of numerical schemes for convection dominated problems. We will discuss the CFL number again when we consider numerical algorithms for the Euler and Navier–Stokes equations.

6.4 Crank–Nicholson Method

In our analysis of the semi-discrete heat equation we have determined that for large N the resulting system of ODE's becomes stiff. Furthermore, the stability requirement when explicit schemes are used is very restrictive. For these reasons, implicit methods are often used for parabolic equations. The most popular implicit method is the second order trapezoidal method, which when applied to PDE's, is called the Crank Nicholson method. Using the trapezoidal method for the heat equation

(6.4) leads to

$$\frac{u^{(n+1)} - u^{(n)}}{\Delta t} = \frac{\nu}{2}\left[\frac{\partial^2 u^{(n+1)}}{\partial x^2} + \frac{\partial^2 u^{(n)}}{\partial x^2}\right] \tag{6.52}$$

Note that this is a semi-discrete equation where we have discretized in time but not in space. If we use the second order central difference for the spatial derivatives then

$$u^{(n+1)} - u^{(n)} = \frac{\nu\Delta t}{2\Delta x^2}\left[u_{j+1}^{(n+1)} - 2u_j^{(n+1)} + u_{j-1}^{(n+1)} + u_{j+1}^{(n)} - 2u_j^{(n)} + u_{j-1}^{(n)}\right] \tag{6.53}$$

If we let $\beta = \nu\Delta t/2\Delta x^2$ and collect terms with superscript $(n+1)$ on the left hand side then we obtain the tri-diagonal system of equations

$$-\beta u_{j+1}^{(n+1)} + (1+2\beta)u_j^{(n+1)} - \beta u_{j-1}^{(n+1)} = \beta u_{j+1}^{(n)} + (1-2\beta)u_j^{(n)} + \beta u_{j-1}^{(n)} \tag{6.54}$$

$$j = 1, 2, 3, \ldots, N-1 \tag{6.55}$$

At every time step, we must solve a tri-diagonal system of equations. In one-dimension this is not too expensive (order N arithmetic operations) however, as we will see below, this process can become very expensive for problems in multiple dimensions.

To verify the unconditional stability of this method let's perform a modified wave number analysis. Recall that the amplification factor for trapezoidal method is

$$\sigma = \frac{1 + \lambda\Delta t/2}{1 - \lambda\Delta t/2} \tag{6.56}$$

From the modified wave number analysis in section 6.3 we know that $\lambda = -\nu k'^2$ where

$$k'^2 = \frac{2}{\Delta x^2}\left[1 - \cos(k\Delta x)\right] \tag{6.57}$$

Thus,

$$\sigma = \frac{1 - \frac{\nu\Delta t}{\Delta x^2}\left[1 - \cos(k\Delta x)\right]}{1 + \frac{\nu\Delta t}{\Delta x^2}\left[1 - \cos(k\Delta x)\right]} \tag{6.58}$$

Since $1 - \cos(k\Delta x) \geq 0$ the denominator is always larger than the numerator, $|\sigma| \leq 1$, and the method is unconditionally stable. However, for large $\nu\Delta t/\Delta x^2$ $\sigma \to -1$ so that the solution oscillates for large Δt but always remains bounded.

6.5 Modified Equation

The numerical solution is only an approximation to the exact solution. Although it satisfies the discretized equations, it does not exactly satisfy the continuous equation. Instead it satisfies a *modified* equation. As an example, consider the heat equation (6.4) and denote $\tilde{u}$ as the exact solution and u as the numerical solution obtained using explicit Euler with second order spatial differencing. The discrete equation can be written as

$$L[u_j^{(n)}] = \frac{u^{(n+1)} - u^{(n)}}{\Delta t} - \nu \frac{u_{j+1}^{(n)} - 2u_j^{(n)} + u_{j-1}^{(n)}}{\Delta x^2} = 0 \tag{6.59}$$

where we have introduced the difference operator $L[u_j^{(n)}]$. To determine the modified equation, every term in (6.59) is expanded in a Taylor series about $u_j^{(n)}$.

$$u_j^{(n+1)} = u_j^{(n)} + \Delta t \left(\frac{\partial u_j}{\partial t}\right)^{(n)} + \frac{\Delta t^2}{2}\left(\frac{\partial^2 u_j}{\partial t^2}\right)^{(n)} + \dots \tag{6.60}$$

Thus,

$$\frac{u^{(n+1)} - u^{(n)}}{\Delta t} = \left(\frac{\partial u_j}{\partial t}\right)^{(n)} + \frac{\Delta t}{2}\left(\frac{\partial^2 u_j}{\partial t^2}\right)^{(n)} + \dots \tag{6.61}$$

Similarly

$$\frac{u_{j+1}^{(n)} - 2u_j^{(n)} + u_{j-1}^{(n)}}{\Delta x^2} = \left(\frac{\partial^2 u^{(n)}}{\partial x^2}\right)_j + \frac{\Delta x^2}{12}\left(\frac{\partial^4 u^{(n)}}{\partial x^4}\right)_j + \dots \tag{6.62}$$

Substituting these expressions into equation (6.59) results in

$$L[u_j^{(n)}] - \left[\left(\frac{\partial u_j}{\partial t}\right)^{(n)} - \nu\left(\frac{\partial^2 u^{(n)}}{\partial x^2}\right)_j\right] = -\nu\frac{\Delta x^2}{12}\left(\frac{\partial^4 u^{(n)}}{\partial x^4}\right)_j + \frac{\Delta t^2}{2}\left(\frac{\partial^2 u_j}{\partial t^2}\right)^{(n)} + \dots \tag{6.63}$$

Since this equation was derived for an arbitrary grid, we can drop the index notation leaving

$$L[u] - \left(\frac{\partial u}{\partial t} - \nu\frac{\partial^2 u}{\partial x^2}\right) = -\nu\frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4} + \frac{\Delta t^2}{2}\frac{\partial^2 u}{\partial t^2} + \dots \tag{6.64}$$

Since u is the solution to the discrete equation, $L[u] = 0$ and we are left with the following *modified* equation which is the continuous equation that the discrete solution satisfies

$$\frac{\partial u}{\partial t} - \nu\frac{\partial^2 u}{\partial x^2} = \nu\frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4} - \frac{\Delta t^2}{2}\frac{\partial^2 u}{\partial t^2} + \dots \tag{6.65}$$

Clearly, as we refine our space-time mesh by letting Δx and Δt approach zero, the modified equation approaches the exact PDE. If we plug the exact solution, $\tilde{u}$, into (6.64) then we obtain

$$L[\tilde{u}] = -\nu \frac{\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4} + \frac{\Delta t^2}{2}\frac{\partial^2 u}{\partial t^2} + \ldots \tag{6.66}$$

and $L[\tilde{u}]$ is known as the truncation error of the method.

Under rather restrictive conditions, it is possible to reduce the truncation error by the judicious selection of Δt and Δx. To see this, lets take $\partial/\partial t$ of (6.4)

$$\frac{\partial^2 \tilde{u}}{\partial t^2} = \nu \frac{\partial^3 \tilde{u}}{\partial t \partial x^2} = \nu^2 \frac{\partial^4 \tilde{u}}{\partial x^4} \tag{6.67}$$

where we have made use of the fact that $\tilde{u}$ is the exact solution to the heat equation. Thus the truncation error can be rewritten as

$$L[\tilde{u}] = \left(-\nu \frac{\Delta x^2}{12} + \nu^2 \frac{\Delta t^2}{2}\right)\frac{\partial^4 u}{\partial x^4} + \ldots \tag{6.68}$$

If we set the term in parentheses equal to zero then we will increase the accuracy of the scheme. Doing so requires that

$$\frac{\nu \Delta t}{\Delta x^2} = \frac{1}{6} \tag{6.69}$$

This value is one-third of the stability limit, making this a stable time step, however, it is a rather small time step and is a very restrictive condition. The reason this works is that the first order errors in space and time compete; the spatial error causes the solution to decay faster than it should, the temporal error makes the solution decay slower than it should. When the time step and mesh size satisfy (6.69) the spatial and temporal errors cancel (to lowest order) leading to a dramatic improvement in accuracy.

6.6 Inconsistency: the Dufort–Frankel Method

Not all numerical methods are consistent. A consistent method is one that when Δx and Δt approach zero, arbitrarily, the modified equation approaches the exact equation. This was certainly the case for the method applied to the heat equation in the previous section.

As an example of an inconsistent method, we consider the Dufort–Frankel method for solving the heat equation. This method is developed in two steps. First, we apply Leapfrog in time and second order central in space to yield

$$\frac{u^{(n+1)} - u^{(n-1)}}{2\Delta t} = \frac{\nu}{\Delta x^2}\left[u_{j+1}^{(n)} - 2u_j^{(n)} + u_{j-1}^{(n)}\right] \tag{6.70}$$

Formally this method is second order in space and time. However, we know from our analysis of the Leapfrog method that it is unstable for real, negative λ. To get a stable scheme, Dufort and Frankel substituted for $u_j^{(n)}$ in the right hand side using the following approximation

$$u_j^{(n)} = \frac{u_j^{(n+1)} + u_j^{(n-1)}}{2} + O(\Delta t^2) \tag{6.71}$$

Since this expression is second order in time, the formal accuracy of the method is unchanged. Rearranging terms leads to

$$(1 + 2\beta)u_j^{(n+1)} = (1 - 2\beta)u_j^{(n-1)} + 2\beta u_{j+1}^{(n)} + 2\beta u_{j-1}^{(n)} \tag{6.72}$$

where $\beta = \nu\Delta t/\Delta x^2$. Applying von Neumann analysis to this equation demonstrates that this method is actually *unconditionally stable*. Thus, we have obtained the holy grail of numerical analysis — an explicit scheme that is always stable. Unfortunately, it is not as good as it seems. If we expand each term in a Taylor series about $u_j^{(n)}$ we obtain the following modified equation

$$\frac{\partial u}{\partial t} - \nu\frac{\partial^2 u}{\partial x^2} = -\frac{\Delta t^2}{6}\frac{\partial^3 u}{\partial t^3} + \frac{\nu\Delta x^2}{12}\frac{\partial^4 u}{\partial x^4} - \frac{\nu\Delta t^2}{\Delta x^2}\frac{\partial^2 u}{\partial t^2} - \frac{\nu\Delta t^4}{12\Delta x^2}\frac{\partial^4 u}{\partial t^4} + \ldots \tag{6.73}$$

The problem with this method lies in the third term on the right hand side. If we hold the time step fixed and refine the spatial mesh, the error actually goes up. Thus, we cannot improve the accuracy of the numerical solution by arbitrarily letting Δx and Δt go to zero. Instead, to make the third term approach zero, we must require that Δt goes to zero faster than Δx. The Dufort–Frankel method is an example of an *inconsistent* numerical method. Consistency is the requirement that the numerical solution approaches the exact solution as Δx and Δt are independently refined.

6.7 Multi-Dimensions

So far, we have only considered one-dimensional problems. Fortunately, the extension to multiple dimensions is relatively straight forward. Let's begin with the two-dimensional heat equation

$$\frac{\partial u}{\partial t} = \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{6.74}$$

and for our numerical solution we introduce a grid in the (x, y) plane with $M + 1$ points in x and $N + 1$ points in y. The boundary points are at $j = 0, M$ or $k = 0, N$ and here we assume that Dirichlet conditions are applied. Let $u_{j,k}^{(n)}$ denote the value of u at the grid point (j, k) and at time step n.

6.7.1 Explicit Schemes

Any explicit time advancement scheme applied to (6.74) leads to a very simple numerical method. For example, applying the explicit Euler method along with second order central differences in space gives

$$\frac{u_{j,k}^{(n+1)} - u_{j,k}^{(n)}}{\Delta t} = \nu \left[\frac{u_{j+1,k}^{(n)} - 2u_{j,k}^{(n)} + u_{j-1,k}^{(n)}}{\Delta x^2} + \frac{u_{j,k+1}^{(n)} - 2u_{j,k}^{(n)} + u_{j,k-1}^{(n)}}{\Delta x^2} \right] \tag{6.75}$$

Starting from an initial condition $u_{j,k}^{(0)}$ one simply marches forward in time to obtain the solution at the next time step. Near the boundaries, $j = 1, M$ or $k = 1, N$ boundary values are required and their values are obtained from the prescribed boundary conditions.

We can use the same methods already discussed for one-dimensional problems to determine the stability properties of the current method. For example, using the modified wave number method, we assume a solution of the form $u = \psi(t)e^{ik_x x + ik_y y}$ and plugging this into the semi-discrete equation (6.75) (discretized in space, not in time) yields

$$\frac{d\psi}{dt} = -\nu \left({k'_x}^2 + {k'_y}^2 \right) \tag{6.76}$$

where the modified wave numbers are given by

$${k'_x}^2 = \frac{2}{\Delta x^2} \left[1 - \cos(k_x \Delta x) \right] \tag{6.77a}$$

$$k_y'^2 = \frac{2}{\Delta y^2}\left[1-\cos(k_y\Delta y)\right] \tag{6.77b}$$

From what we know of the explicit Euler method, the following condition must be satisfied for a stable solution

$$\Delta t \le \frac{2}{\nu\left\{\frac{2}{\Delta x^2}\left[1-\cos(k_x\Delta x)\right]+\frac{2}{\Delta y^2}\left[1-\cos(k_y\Delta y)\right]\right\}} \tag{6.78}$$

The worst case occurs when $\cos(k_x\Delta x) = -1$ and $\cos(k_y\Delta y) = -1$, thus

$$\Delta t \le \frac{1}{2\nu\left\{\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}\right\}} \tag{6.79}$$

When the mesh is square, $\Delta x = \Delta y = h$ and we have the special case

$$\Delta t \le \frac{h^2}{4\nu} \tag{6.80}$$

which is two times more restrictive than the one-dimensional case. In three-dimensions for a square mesh, one obtains

$$\Delta t \le \frac{h^2}{6\nu} \tag{6.81}$$

Even in one-dimension, explicit schemes are not often practical for parabolic equations. We see here that the situation is even worse in multiple dimensions.

6.7.2 Implicit Schemes

Implicit schemes are often the methods of choice to overcome the severe time step restrictions associated with explicit schemes for parabolic problems. If we apply Crank–Nicholson (*i.e.*trapezoidal method) to the 2-D heat equations we obtain

$$\frac{u^{(n+1)}-u^{(n)}}{\Delta t} = \frac{\nu}{2}\left[\frac{\partial^2 u^{(n+1)}}{\partial x^2}+\frac{\partial^2 u^{(n+1)}}{\partial y^2}+\frac{\partial^2 u^{(n)}}{\partial x^2}+\frac{\partial^2 u^{(n)}}{\partial y^2}\right] \tag{6.82}$$

If we use second order central in space, and for simplicity, assume a square mesh then we obtain, after some rearrangement,

$$\begin{aligned} u_{j,k}^{(n+1)} - u_{j,k}^{(n)} &= \frac{\nu\Delta t}{2h^2}\left[u_{j+1,k}^{(n+1)} - 2u_{j,k}^{(n+1)} + u_{j-1,k}^{(n+1)} + u_{j,k+1}^{(n+1)} - 2u_{j,k}^{(n+1)} + u_{j,k-1}^{(n+1)}\right] \\ &+ \frac{\nu\Delta t}{2h^2}\left[u_{j+1,k}^{(n)} - 2u_{j,k}^{(n)} + u_{j-1,k}^{(n)} + u_{j,k+1}^{(n)} - 2u_{j,k}^{(n)} + u_{j,k-1}^{(n)}\right] \end{aligned} \tag{6.83}$$

If we let $\beta = \nu\Delta t/2h^2$ and collect unknowns on the left hand side, then

$$\begin{aligned}-\beta u_{j,k}^{(n+1)} + (1+4\beta)u_{j,k}^{(n+1)} - \beta u_{j-1,k}^{(n+1)} - \beta u_{j,k+1}^{(n+1)} - \beta u_{j,k-1}^{(n+1)} = \\ \beta u_{j,k}^{(n)} + (1-4\beta)u_{j,k}^{(n)} + \beta u_{j-1,k}^{(n)} + \beta u_{j,k+1}^{(n)} + \beta u_{j,k-1}^{(n)} = F^{(n)}\end{aligned} \tag{6.84}$$

which is a system of linear algebraic equations for $u_{j,k}^{(n+1)}$. Writing this system in matrix form results in the following $[(M-1)(N-1)] \times [(M-1)(N-1)]$ block-tridiagonal matrix

$$\mathbf{G} = \begin{bmatrix} \mathbf{B} & \mathbf{C} & & & \\ \mathbf{A} & \mathbf{B} & \mathbf{C} & & \\ & \ddots & \ddots & \ddots & \\ & & \mathbf{A} & \mathbf{B} & \mathbf{C} \\ & & & \mathbf{A} & \mathbf{B} \end{bmatrix} \tag{6.85}$$

where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are $(M-1)\times(M-1)$ matrices. In this case, $\mathbf{A}$ and $\mathbf{C}$ are diagonal matrices while $\mathbf{B}$ is tri-diagonal. For practical problems $\mathbf{G}$ can become *very* large. For example, if $M = 101$ and $N = 101$ then $\mathbf{G}$ has 10^8 elements. However, it we take advantage of the banded structure of the matrix $\mathbf{G}$ in the Gauss elimination process we will only need to store $2(M-1)^2N$ elements. For the current example this amounts to 2×10^6 words of memory. To put this in perspective, if we are using 8 byte floating point numbers then the matrix $\mathbf{G}$ requires 16 mega-bytes of memory to store and solve!

6.7.3 Approximate Factorization

There have been a variety of ways that have been developed to reduce the storage requirements of implicit methods applied to multi-dimensional problems. These methods include iterative solution techniques, described in the next section, and split or factored schemes. The advantage of factored schemes is that they avoid large matrices and iteration while maintaining the same order of accuracy. Consider again the two-dimensional heat equation (6.74) with Crank–Nicholson in time and second order central in space. The discrete equation is given by (6.75), but for the purpose at hand the following form is more useful

$$\begin{aligned}\frac{u^{(n+1)} - u^{(n)}}{\Delta t} = \frac{\nu}{2}\boldsymbol{\delta}_x \left[u^{(n+1)} + u^{(n)}\right] + \frac{\nu}{2}\boldsymbol{\delta}_y \left[u^{(n+1)} + u^{(n)}\right] + \\ O(\Delta t^2) + O(\Delta x^2) + O(\Delta y^2)\end{aligned} \tag{6.86}$$

where $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$ are shorthand notation for the finite difference operators in x and y respectively. For example, $\boldsymbol{\delta}_x u$ is a vector of length $(N-1)(M-1)$ with the following elements

$$\frac{u_{j+1,k} - 2u_{j,k} + u_{j-1,k}}{\Delta x^2} \quad j = 1, 2, \ldots, (M-1) \quad k = 1, 2, \ldots, (N-1) \tag{6.87}$$

If we move terms involving time step $n+1$ onto the left hand side, and those at n on the right hand side then

$$\begin{aligned}\left[\mathbf{I} - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_x - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_y\right] u^{(n+1)} = \left[\mathbf{I} + \frac{\nu\Delta t}{2}\boldsymbol{\delta}_x + \frac{\nu\Delta t}{2}\boldsymbol{\delta}_y\right] u^{(n)} +& \\ \Delta t\left[O(\Delta t^2) + O(\Delta x^2) + O(\Delta y^2)\right]&\end{aligned} \tag{6.88}$$

We can then factor the terms in square brackets

$$\begin{aligned}\left(\mathbf{I} - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_x\right)\left(\mathbf{I} - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_y\right) u^{(n+1)} - \frac{\nu^2\Delta t^2}{4}\boldsymbol{\delta}_x\boldsymbol{\delta}_y u^{(n+1)} =& \\ \left(\mathbf{I} + \frac{\nu\Delta t}{2}\boldsymbol{\delta}_x\right)\left(\mathbf{I} + \frac{\nu\Delta t}{2}\boldsymbol{\delta}_y\right) u^{(n)} - \frac{\nu^2\Delta t^2}{4}\boldsymbol{\delta}_x\boldsymbol{\delta}_y u^{(n)} +& \\ \Delta t\left[O(\Delta t^2) + O(\Delta x^2) + O(\Delta y^2)\right]&\end{aligned} \tag{6.89}$$

and rewrite this as

$$\begin{aligned}\left(\mathbf{I} - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_x\right)\left(\mathbf{I} - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_y\right) u^{(n+1)} = \left(\mathbf{I} + \frac{\nu\Delta t}{2}\boldsymbol{\delta}_x\right)\left(\mathbf{I} + \frac{\nu\Delta t}{2}\boldsymbol{\delta}_y\right) u^{(n)}& \\ + \frac{\nu^2\Delta t^2}{4}\left(u^{(n+1)} - u^{(n)}\right) + \Delta t\left[O(\Delta t^2) + O(\Delta x^2) + O(\Delta y^2)\right]&\end{aligned} \tag{6.90}$$

However, from a Taylor series expansion we know that $u^{(n+1)} - u^{(n)} = O(\Delta t)$, so that the cross-derivative term is of the same order of accuracy as the truncation error and can therefore be neglected. Thus we arrive at the approximately factored form of the discrete equations

$$\left(\mathbf{I} - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_x\right)\left(\mathbf{I} - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_y\right) u^{(n+1)} = \left(\mathbf{I} + \frac{\nu\Delta t}{2}\boldsymbol{\delta}_x\right)\left(\mathbf{I} + \frac{\nu\Delta t}{2}\boldsymbol{\delta}_y\right) u^{(n)} \tag{6.91}$$

which we will now show to be considerably more memory efficient to solve then the unfactored equation.

To solve the factored equation (6.91) we first denote the known right hand side by f. Furthermore, if we let

$$\left(\mathbf{I} - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_y\right) u^{(n+1)} = z \tag{6.92}$$

then we can solve for z from the following equation

$$\left(\mathbf{I} - \frac{\nu\Delta t}{2}\boldsymbol{\delta}_x\right) z = f \tag{6.93}$$

In index form this equation becomes

$$-\frac{\nu\Delta t}{2\Delta x^2} z_{j+1,k} + \left(1 + \frac{\nu\Delta t}{\Delta x^2}\right) z_{j,k} - \frac{\nu\Delta t}{2\Delta x^2} z_{j-1,k} = f_{j,k} \tag{6.94}$$

Thus, for each value of $k = 1, 2, \ldots, (N-1)$, a tri-diagonal system of equations must be solved for $z_{i,j}$. Once z is know, we can solve for $u^{(n+1)}$ from (6.92) which, in index notation, is

$$-\frac{\nu\Delta t}{2\Delta y^2} u_{j,k+1}^{(n+1)} + \left(1 + \frac{\nu\Delta t}{\Delta y^2}\right) u_{j,k}^{(n+1)} - \frac{\nu\Delta t}{2\Delta y^2} u_{j,k-1}^{(n+1)} = z_{j,k} \tag{6.95}$$

which is also a tri-diagonal system of equations that must be solved for each value of $i = 1, 2, \ldots, (M-1)$.

Instead of solving a single large system of size $(M-1)^2 \times (N-1)^2$, we now solve $(M-1)$ tri-diagonal systems of size $(N-1)$ and then $(N-1)$ tri-diagonal systems of size $(M-1)$. Both the number of operations and the number of words of memory required is on the order of MN.

The reader may have noticed that to solve the system of equations (6.94) requires boundary conditions for z at $j = 1, N$. However, in the physical problem, boundary conditions are only prescribed for u. To generate boundary conditions for z one can use equation (6.92). As an example, consider the boundary at $x = 0$. Here, $z_{0,j}$ is computed from

$$z_{0,k} = u_{0,k}^{(n+1)} - \frac{\nu\Delta t}{2\Delta y^2}\left(u_{0,k+1}^{(n+1)} - 2u_{0,k}^{(n+1)} + u_{0,k-1}^{(n+1)}\right) \tag{6.96}$$

Note that the boundary data are at time-step $(n+1)$. An equation similar to this is obtained at the other boundaries. Note that we have factored the original equation such that we first solve systems in x and then solve systems in y. We could just as easily have factored the original problem such that we solved first in y then in x. In practice, one can alternate between the two approaches to reduce anisotropy in the numerical method.

Using the modified wave number analysis or von Neumann's method, it can be shown that the factored discrete equation retains unconditional stability in this case. However, this is not always the case, especially if higher-order finite difference formulas are used for the spatial derivatives. Also, approximate factorization applied in three-dimensions leads to only conditional stability even for second order accurate schemes. It is important to note, while the formal order of accuracy of the factor form is the same as the original equation, the actual level of error will likely be higher due to the neglect of the cross derivative term.

6.8 Iterative Methods

In many instances, especially for high-order methods and methods on unstructured meshes, it is impractical to solve the resulting system of linear equations using a direct method or even approximate factorization. In these cases, one must resort to iterative solution methods that generate an approximate solution to the linear system of equations at much less expense. By far the most developed theory and methods for iterative solution are for elliptic PDE's. Thus we begin this section with a brief review of elliptic equations. We then show that the numerical solution of elliptic PDE naturally leads to a large system of algebraic equations that are typically too large for direct solution methods. We then introduce a range of iterative methods for the solution of these linear systems. It is important to remember that the iterative methods presented are typically valid for general linear systems of equations – we just use the numerical solution of elliptical PDE as motivation.

6.8.1 Elliptic Partial Differential Equations

The classical example of an elliptic PDE is Laplace's equation which can be written as

$$\nabla^2\phi = 0 \tag{6.97}$$

where the form of ∇^2 depends on the coordinate system in use. For example, in two-dimensional Cartesian coordinates

$$\nabla^2\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} \tag{6.98}$$

A source term is added to the right hand side of the Laplace equation leads to the Poisson equation

$$\nabla^2\phi = f \tag{6.99}$$

If a time derivative term is added to the Laplace equation and a Fourier decomposition is used in time, then we obtain an equation of the form

$$\nabla^2\phi + \omega\phi = 0 \tag{6.100}$$

which is called a Helmholtz equation.

As an example of a numerical solution of an elliptical PDE, consider the application of second order central differences to the two-dimensional Poisson equation in Cartesian coordinates

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \tag{6.101}$$

which can be rewritten as

$$\phi_{i+1,j} - 4\phi_{i,j} + \phi_{i-1,j} + \phi_{i,j-1} + \phi_{i,j+1} = h^2 f_{i,j} \tag{6.102}$$

for $i = 1, 2, \ldots, M-1$ and $j = 1, 2, \ldots, N-1$. If we index the unknown $\phi_{i,j}$ first with i then with j, we have the vector of unknowns

$$\mathbf{x} = \{\phi_{1,1}, \phi_{2,1}, \phi_{3,1}, \ldots, \phi_{M-1,1}, \phi_{1,2}, \phi_{2,2}, \ldots, \phi_{M-1,N-1}\}^T \tag{6.103}$$

Then we can rewrite the systems of equations in the matrix form

$$\mathbf{Ax} = \mathbf{b} \tag{6.104}$$

The matrix $\mathbf{A}$ is block tri-diagonal where each block is $(M-1) \times (M-1)$ and there are $(N-1)$ blocks on the main diagonal. Different discretizations lead to different banded matrices. For example, fourth order central differencing will result in a block penta-diagonal matrix. For the second order discretization used here, efficient direct solution techniques have been developed. However, for complex geometries with non-uniform meshes, the matrix structure may not be amiable to direct solution. In these cases one often uses an iterative method to obtain an approximate solution.

6.8.2 Iterative methods

In the following sections we consider the solution of (6.104) using iterative methods. It is important to remember that the system of algebraic equations is general — it may or may not have been derived from the numerical solution to a PDE. To derive our iterative method, let $\mathbf{A} = \mathbf{A}_1 - \mathbf{A}_2$. Then equation (6.104) can be written as

$$\mathbf{A}_1\mathbf{x} = \mathbf{A}_2\mathbf{x} + \mathbf{b} \tag{6.105}$$

and an iterative method is obtained by writing

$$\mathbf{A}_1\mathbf{x}^{(k+1)} = \mathbf{A}_2\mathbf{x}^{(k)} + \mathbf{b} \tag{6.106}$$

where $k = 1, 2, 3, \ldots$ is the iteration index. In order to obtain a useful iterative method, the following two conditions must be met

1. The matrix $\mathbf{A}_1$ must be easy to solve. Otherwise we might as well solve the original system.
2. The iterations must converge (hopefully rapidly). We can state this mathematically as

$$\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x} \tag{6.107}$$

To determine convergence, let's define the error at the k^{th} iteration as

$$\boldsymbol{\epsilon}^{(k)} = \mathbf{x} - \mathbf{x}^{(k)} \tag{6.108}$$

If we subtract (6.105) from (6.106) then

$$\mathbf{A}_1\boldsymbol{\epsilon}^{(k+1)} = \mathbf{A}_2\boldsymbol{\epsilon}^{(k)} \tag{6.109}$$

or, upon solving for the new error

$$\boldsymbol{\epsilon}^{(k+1)} = \mathbf{A}_1^{-1}\mathbf{A}_2\boldsymbol{\epsilon}^{(k)} \tag{6.110}$$

We can use this expression to relate the error at k to the initial error

$$\boldsymbol{\epsilon}^{(k)} = \left(\mathbf{A}_1^{-1}\mathbf{A}_2\right)^k \boldsymbol{\epsilon}^{(0)} \tag{6.111}$$

and for convergence we must have

$$\lim_{k\to\infty} \boldsymbol{\epsilon}^{(k)} = 0 \tag{6.112}$$

We know from linear algebra that the powers of a matrix for large exponents go to zero if the modulus of its eigenvalues are all less than 1. Thus the condition for convergence is that

$$\rho = |\lambda_i|_{max} \leq 1 \tag{6.113}$$

where λ_i are the eigenvalues of the matrix $\mathbf{A}_1^{-1}\mathbf{A}_2$, and ρ is the called the spectral radius of convergence for the iterative method. The rate of convergence, and hence the performance, of any iterative method applied to a matrix $\mathbf{A}$ depends on how it is decomposed into $\mathbf{A}_1$ and $\mathbf{A}_2$.

6.8.3 Point Jacobi Method

The very simplest choice for the matrix $\mathbf{A}_1$ is the diagonal matrix which consists of the diagonal elements of $\mathbf{A}$. For the numerical solution of the Poisson equation in equation (6.102), $\mathbf{A}_1$ is the diagonal matrix with -4 on the diagonal. The inverse $\mathbf{A}_1^{-1}$ is just a diagonal matrix with $-1/4$ on the diagonal. The matrix $\mathbf{A}_2$ is easily derived by removing the diagonal and multiplying by -1. Thus, if we apply point Jacobi to equation (6.102) then we obtain the iterative scheme

$$\mathbf{x}^{(k+1)} = -\frac{1}{4}\mathbf{A}_2\mathbf{x}^{(k)} - \frac{1}{4}\mathbf{b} \tag{6.114}$$

Using the spatial index notation, this is just

$$\phi_{i,j}^{(k+1)} = \frac{1}{4}\left[\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k)} + \phi_{i,j-1}^{(k)} + \phi_{i,j+1}^{(k)}\right] - \frac{1}{4}b_{i,j} \tag{6.115}$$

Starting with an initial guess $\phi_{i,j}^{(0)}$ we can easily compute improved approximations using equation (6.115). There is no matrix to solve; the updated value is simply computed using an average of the surrounding points from the previous iteration.

The convergence of this method depends on the eigenvalues of the matrix $-\frac{1}{4}\mathbf{A}_2$ which in this simple case are given by

$$\lambda_{mn} = \frac{1}{2}\left[\cos\frac{m\pi}{M} + \cos\frac{n\pi}{N}\right] \tag{6.116}$$

$$m = 1, 2, 3, \ldots, (M-1) \qquad n = 1, 2, 3, \ldots, (N-1) \tag{6.117}$$

Clearly the method converges, since $|\lambda_{mn}| \leq 1$ for all m and n. The largest eigenvalue magnitude determines the convergence rate. If we expand (6.116) for large M and N we get

$$|\lambda|_{max} = 1 - \frac{1}{4}\left[\frac{\pi^2}{M^2} + \frac{\pi^2}{N^2}\right] + \ldots \tag{6.118}$$

Thus, for large systems, $|\lambda|_{max}$ is only slightly less than 1 and the convergence of this method can be very slow indeed. It is worth emphasizing that the matrix form of the iterative method (6.114) is useful for analysis *but should not be used to implement the method.* One should always develop an expression such as (6.115) that is used in coding the method on a computer.

6.8.4 Gauss-Seidel Method

To improve upon the convergence of the point Jacobi method for our Poisson problem, we can realize that both $\phi_{i-1,j}^{(k+1)}$ and $\phi_{i,j-1}^{(k+1)}$ are computed from equation (6.115) before $\phi_{i,j}^{(k+1)}$. If instead of

$\phi_{i-1,j}^{(k)}$ and $\phi_{i,j-1}^{(k)}$ we use their updated values, which are presumably more accurate, then we obtain the formula for the Gauss–Seidel method

$$\phi_{i,j}^{(k+1)} = \frac{1}{4}\left[\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k+1)} + \phi_{i,j-1}^{(k+1)} + \phi_{i,j+1}^{(k)}\right] - \frac{1}{4}b_{i,j} \tag{6.119}$$

This is the equation that is used to code the Gauss–Seidel method for our example problem. In order to analyze the convergence properties of Gauss–Seidel, we can write the method in terms of the matrix decomposition

$$\mathbf{A} = \mathbf{A}_1 - \mathbf{A}_2 \tag{6.120}$$

where

$$\mathbf{A}_1 = \mathbf{D} - \mathbf{L} \qquad \text{and} \qquad \mathbf{A}_2 = \mathbf{U} \tag{6.121}$$

Here, $\mathbf{D}$ is the diagonal and $\mathbf{L}$ is the negative of the lower triangular part of $\mathbf{A}$. Likewise, $\mathbf{U}$ is the negative of the upper triangular part of $\mathbf{A}$. Note that $\mathbf{L}$ and $\mathbf{U}$ are not to be confused with the usual $\mathbf{LU}$ decomposition of $\mathbf{A}$. Since $\mathbf{A}_1$ is lower triangular, it is easy to solve. In addition, it turns out that for the problem under consideration, the eigenvalues of $\mathbf{A}_1^{-1}\mathbf{A}_2$ are just the square of the point Jacobi eigenvalues

$$\lambda_{mn} = \frac{1}{4}\left[\cos\frac{m\pi}{M} + \cos\frac{n\pi}{N}\right]^2 \tag{6.122}$$

$$m = 1, 2, 3, \ldots, (M-1) \qquad n = 1, 2, 3, \ldots, (N-1) \tag{6.123}$$

Thus the Gauss–Seidel method converges twice as fast as point Jacobi. In more precise terms, Gauss–Seidel requires half the number of iterations as point Jacobi to reduce the error to within a certain tolerance.

6.8.5 Successive Over Relaxation (SOR)

We can further improve upon the Gauss–Seidel method by introducing the concept of Successive Over Relaxation (SOR). To analyze this method, let's begin with Gauss–Seidel in the form

$$(\mathbf{D} - \mathbf{L})\mathbf{x}^{(k+1)} = \mathbf{U}\mathbf{x}^{(k)} + \mathbf{b} \tag{6.124}$$

If we denote the change in the solution between two consecutive iterations as

$$\mathbf{d} = \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \tag{6.125}$$

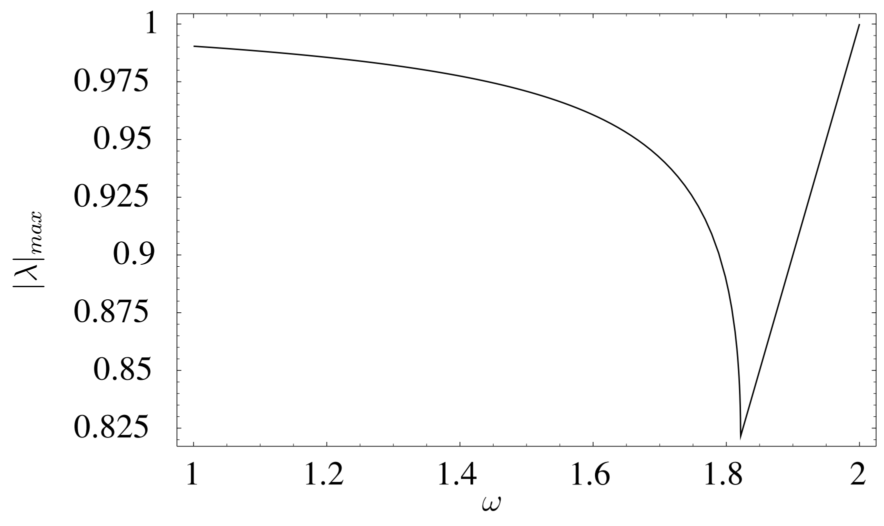

Figure 6.3: Maximum eigenvalue for SOR applied to the Poisson equation as a function of the acceleration parameter, ω. These results are for a (32×32) two-dimensional, square mesh using second order central differencing.

then for Gauss–Seidel we simply have

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d} \tag{6.126}$$

The idea behind SOR is that we can attempt to accelerate convergence by introducing the parameter ω such that

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega\mathbf{d} \tag{6.127}$$

where $\omega > 1$ is called the acceleration parameter. Thus in SOR we find an intermediate solution, $\tilde{\mathbf{x}}^{(k+1)}$

$$\mathbf{D}\tilde{\mathbf{x}}^{(k+1)} = \mathbf{L}\mathbf{x}^{(k+1)} + \mathbf{U}\mathbf{x}^{(k)} + \mathbf{b} \tag{6.128}$$

and then the solution at the next iteration is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega(\tilde{\mathbf{x}}^{(k+1)} - \mathbf{x}^{(k)}) \tag{6.129}$$

Notice that the SOR method is just a Gauss–Seidel predictor followed by an accelerated correction to get the solution at $k+1$. Applying SOR to our example problem, leads to

$$\tilde{\phi}_{i,j}^{(k+1)} = \frac{1}{4}\left[\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k+1)} + \phi_{i,j-1}^{(k+1)} + \phi_{i,j+1}^{(k)}\right] - \frac{1}{4}b_{i,j} \tag{6.130}$$

$$\phi_{i,j}^{(k+1)} = \phi_{i,j}^{(k)} + \omega(\tilde{\phi}_{i,j}^{(k+1)} - \phi_{i,j}^{(k)}) \tag{6.131}$$

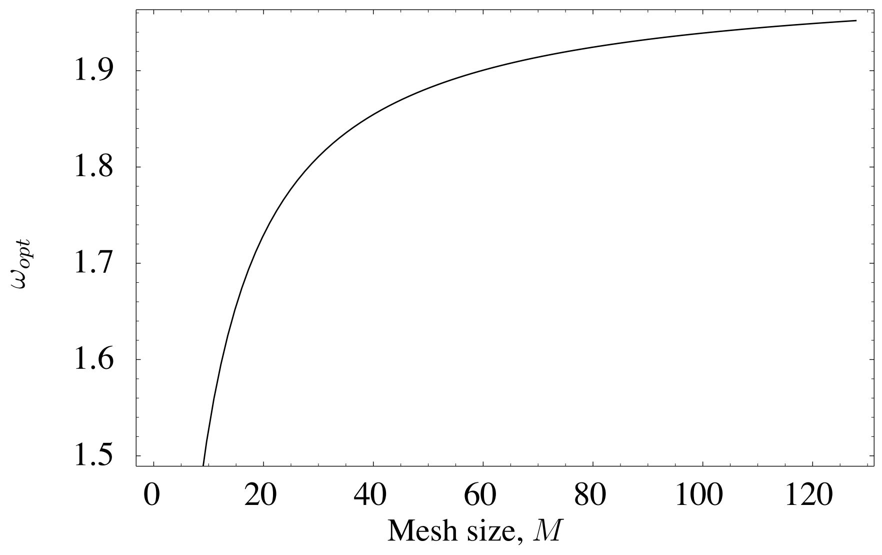

Figure 6.4: Optimal acceleration parameter, ω_{opt} for a square $M \times M$ Cartesian mesh.

which is used for all interior points.

To study the convergence properties of this solution we eliminate the intermediate solution and obtain

$$\mathbf{x}^{(k+1)} = \underbrace{(\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{L})^{-1}\left[(1-\omega)\mathbf{I} + \omega\mathbf{D}^{-1}\mathbf{U}\right]}_{\mathbf{G}_{SOR}}\mathbf{x}^{(k)} + (\mathbf{I} - \omega\mathbf{D}^{-1}\mathbf{L})^{-1}\omega\mathbf{D}^{-1}\mathbf{b} \tag{6.132}$$

For convergence, the eigenvalues of the matrix $\mathbf{G}_{SOR}$ must have magnitude less than one. For the discretized Poisson equation that we have been studying, the eigenvalue are given by

$$\lambda = \frac{1}{4}\left[\mu\omega + \sqrt{\mu^2\omega^2 - 4(\omega - 1)}\right]^2 \tag{6.133}$$

where μ is an eigenvalue of the point Jacobi method, $\mathbf{G}_{PJ} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$.

In SOR we have the ability to optimize convergence by selecting the value of ω such that $|\lambda|$ is minimized. It turns out that there is not a point where $d\lambda/(d)\omega = 0$ but there is an absolute minimum in a plot of λ vs. ω. An example of such a plot is shown in figure 6.3 for a square mesh with $M = N = 32$. From the figure we see that the minimum occurs at $\omega \approx 1.82$. Note that $d\lambda/d\omega$ is not defined at this value of ω since this is the location of a branch point (*i.e.* λ becomes complex for $\omega > \omega_{opt}$). One can derive an expression for the optimal ω for this problem

$$\omega_{opt} = \frac{2}{1 + \sqrt{1 - \mu_{max}^2}} \tag{6.134}$$

Recall from our analysis of point Jacobi, that μ_{max} is just slightly less than 1. Figure 6.4 shows the variation of ω_{opt} with the problem size assuming a square mesh. For typical problems, the optimum value for ω usually lies between 1.7 and 1.95. A closed form expression for ω_{opt}, such as equation (6.134), is usually not available and for more complicated systems of equations (*i.e.* on a nonuniform mesh) one must find the optimal value of ω through numerical experimentation. A good initial guess for ω is 1.8 and experimentation can be used to refine this value if needed.

6.8.6 Alternating Direction Implicit

The Alternating Direction Implicit (ADI) method was originally developed as a method for solving the unsteady heat equation [4] but was later adapted to be used as an iterative method for solving elliptic problems [2, 3]. By way of example, consider the Poisson equation

$$\nabla^2 u + f(x, y) = 0$$

Let's introduce an artificial time, τ, such that

$$\frac{\partial u}{\partial \tau} = \nabla^2 u + f$$

Then, using implicit Euler in τ and second order central in space

$$\frac{u^{(n+1)} - u^{(n)}}{\Delta\tau} - (\boldsymbol{\delta}_x + \boldsymbol{\delta}_y)u^{(n+1)} = f$$

where $\boldsymbol{\delta}_x$ and $\boldsymbol{\delta}_y$ are the finite difference operators. This equation can be rewritten as

$$\left[\mathbf{I} - \Delta\tau\boldsymbol{\delta}_x - \Delta\tau\boldsymbol{\delta}_y\right] u^{(n+1)} = u^{(n)} + \Delta\tau f$$

It is often useful in iterative methods to solve for the correction, $\Delta u^{(n)} = u^{(n+1)} - u^{(n)}$, instead of solving directly for the solution, $u^{(n+1)}$. In this case, we have

$$\left[\mathbf{I} - \Delta\tau\boldsymbol{\delta}_x - \Delta\tau\boldsymbol{\delta}_y\right] \Delta u^{(n)} = \Delta\tau \left[(\boldsymbol{\delta}_x + \boldsymbol{\delta}_y)u^{(n)} + f\right]$$

where the right-hand-side is just $\Delta\tau$ times the residual of the original equation evaluated at n. When the iteration converges, the residual will be identically zero. Using approximate factorization leads to

$$\left(\mathbf{I} - \Delta\tau\boldsymbol{\delta}_x\right)\left(\mathbf{I} - \Delta\tau\boldsymbol{\delta}_y\right) \Delta u^{(n)} = \Delta\tau \left[(\boldsymbol{\delta}_x + \boldsymbol{\delta}_y)u^{(n)} + f\right]$$

Thus we can write the ADI scheme as

$$(\mathbf{I} - \Delta\tau\boldsymbol{\delta}_x)\,\overline{\Delta u^{(n)}} = \Delta\tau\left[(\boldsymbol{\delta}_x + \boldsymbol{\delta}_y)u^{(n)} + f\right]$$

$$(\mathbf{I} - \Delta\tau\boldsymbol{\delta}_y)\,\Delta u^{(n)} = \overline{\Delta u^{(n)}}$$

As for the heat equation, we have converted a large two-dimensional system of equations into a series of one-dimensional tri-diagonal systems of equations in x and y.

6.9 Methods for Hyperbolic Equations

In the previous sections, we have used the one-dimensional wave equation

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0 \tag{6.135}$$

as a model for hyperbolic PDEs. As a generalization of this model, we introduce the generic one-dimensional conservation law of the form

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \tag{6.136}$$

where $f(u)$ is the flux function for u-stuff in the x-direction. Note that usually f is a nonlinear function of u. When the spatial derivative is written in the form of the divergence of a flux function, then we say that the equation is in *conservative* form. Starting from (6.136) we can construct the corresponding nonconservative form by using the chain-rule for differentiation

$$\frac{\partial u}{\partial t} + a(u)\frac{\partial u}{\partial x} = 0 \tag{6.137}$$

where $a(u) = \partial f/\partial u$. If instead of a scalar equation, we have a vector conservation law

$$\frac{\partial \vec{u}}{\partial t} + \frac{\partial \vec{f}}{\partial x} = 0 \tag{6.138}$$

Then the nonconservative form is

$$\frac{\partial \vec{u}}{\partial t} + \boldsymbol{A}(\vec{u})\frac{\partial \vec{u}}{\partial x} = 0 \tag{6.139}$$

where $A_{ij} = \partial f_i/\partial u_j$ are the components of the Jacobian matrix, $\boldsymbol{A}$.

6.9.1 Lax-Fredrich Method

The Lax-Fredrich method starts with explicit Euler in time and second-order central in space.

$$u_i^{(n+1)} = u_i^{(n)} - \frac{\Delta t}{2\Delta x}\left(f_{i+1}^{(n)} - f_{i-1}^{(n)}\right) \tag{6.140}$$

From the modified wave number analysis, we know that this method is unstable. To correct for this, we approximate $u_i^{(n)}$ with

$$u_i^{(n)} \approx \frac{u_{i+1}^{(n)} + u_{i-1}^{(n)}}{2} \tag{6.141}$$

which yields

$$u_i^{(n+1)} = \frac{u_{i+1}^{(n)} + u_{i-1}^{(n)}}{2} - \frac{\Delta t}{2\Delta x}\left(f_{i+1}^{(n)} - f_{i-1}^{(n)}\right) \tag{6.142}$$

This method is $\mathcal{O}(\Delta x^2, \Delta t)$ and it is easy to show with a von Neumann analysis that it is linearly stable for $CFL \leq 1$. For $CFL = 1$ the spatial and temporal errors cancel given a nodally exact method for linear constant coefficient equations.

6.9.2 Lax-Wendroff Method

The Lax-Wendroff method is a second-order space-time method that is based upon a central difference for the spatial derivatives with a second-order Taylor series method in time. Let's begin with a Taylor series expansion of the form

$$u^{(n+1)} = u^{(n)} + \Delta t \left.\frac{\partial u}{\partial t}\right|_n + \Delta t^2 \left.\frac{\partial^2 u}{\partial t^2}\right|_n + \mathcal{O}(\Delta t^3) \tag{6.143}$$

From the PDE we know that

$$\frac{\partial u}{\partial t} = -\frac{\partial f}{\partial x} \tag{6.144}$$

and

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial u}\frac{\partial f}{\partial x}\right) = \frac{\partial}{\partial x}\left(a(u)\frac{\partial f}{\partial x}\right) \tag{6.145}$$

so that for a globally second-order method we have

$$u^{(n+1)} = u^{(n)} - \Delta t \left.\frac{\partial f}{\partial x}\right|_n + \Delta t^2 \frac{\partial}{\partial x}\left.\left(a(u)\frac{\partial f}{\partial x}\right)\right|_n \tag{6.146}$$

The method is completed by using second-order central difference operators leading to a fully discrete method given by

$$u_i^{(n+1)} = u_i^{(n)} - \frac{\Delta t}{2\Delta x}\left(f_{i+1}^{(n)} - f_{i-1}^{(n)}\right) + \frac{\Delta t^2}{\Delta x^2}\left[a_{i+1/2}^{(n)}(f_{i+1}^{(n)} - f_i^{(n)}) - a_{i-1/2}^{(n)}(f_i^{(n)} - f_{i-1}^{(n)})\right] \quad (6.147)$$

where $a_{i+1/2}^{(n)} = (a_i^{(n)}+a_{i+1}^{(n)})/2$ and $a_{i-1/2}^{(n)} = (a_i^{(n)}+a_{i-1}^{(n)})/2$. For systems of nonlinear equations the Jacobian of the flux function, $\boldsymbol{A}(\vec{u})$ is required which makes this method somewhat inconvenient. However, for the constant coefficient linear wave equation the method simply becomes

$$u^{(n+1)} = u^{(n)} - c\Delta t\frac{\partial u}{\partial x}\bigg|_n + c^2\Delta t^2\frac{\partial^2 u}{\partial x^2}\bigg|_n \quad (6.148)$$

with second-order first and second derivatives approximations in space. The method is $\mathcal{O}(\Delta t^2, \Delta^2)$ and linearly stable for $CFL \leq 1$. Thus, this is the first explicit second-order accurate method that we have introduced for hyperbolic equations.

6.9.3 Richtmyer Method

The Richtmyer method addresses the limitation of the Lax-Wendroff method for nonlinear equations. The method is given by a Lax-Fredrich predictor followed by a Leapfrog/Central corrector. For the model problem this yields

$$u_i^{(n+1)} = \frac{u_{i+1/2}^{(n)} + u_{i-1}^{(n)}}{2} - \frac{\Delta t}{4\Delta x}\left(f_{i+1}^{(n)} - f_{i-1}^{(n)}\right) \quad (6.149a)$$

$$u_i^{(n+1)} = u_i^{(n)} - \frac{\Delta t}{2\Delta x}\left(f_{i+1}^{(n+1/2)} - f_{i-1}^{(n+1/2)}\right) \quad (6.149b)$$

The reader should show that this method is $\mathcal{O}(\Delta t^2, \Delta x^2)$ and is stable for $CFL \leq 1$. For linear problems this method is identical to the Lax-Wendroff method.

6.9.4 MacCormack Method

One of the most popular explicit methods in CFD is the MacCormack method, developed by Robert MacCormack in the early 1970's. This method is simple, second-order accurate in space and time, and is fully explicit. And, like the Richtmyer method, MacCormack's method does not require Jacobians.

It is easiest to demonstrate the method through an example by applying it to the model equation

(6.136) as follows

$$\overline{u}_i = u_i^{(n)} - \frac{\Delta t}{\Delta x}\left(f_{i+1}^{(n)} - f_i^{(n)}\right) \qquad \text{Predictor} \tag{6.150a}$$

$$u_i^{(n+1)} = \frac{1}{2}\left[u_i^{(n)} + \bar{u}_i - \frac{\Delta t}{\Delta x}\left(\overline{f}_i - \overline{f}_{i-1}\right)\right] \qquad \text{Corrector} \tag{6.150b}$$

which can be interpreted as an explicit Euler predictor with forward difference and an explicit Euler corrector with backward difference. Assuming that information propagates in the $+x$-direction, the predictor stage is known to be unstable, while the corrector stage is stable, but over diffuse. MacCormacks method amounts to the average of these two methods where the result is a second order accurate method that is stable for $CFL \leq 1$. For linear problems with $CFL = 1$ the solution is nodally exact since the errors in space and time exactly cancel. MacCormack's method can also be written in the following symmetric form

$$\overline{u}_i = u_i^{(n)} - \frac{\Delta t}{\Delta x}\left(f_{i+1}^{(n)} - f_i^{(n)}\right) \qquad \text{Predictor} \tag{6.151a}$$

$$\overline{\overline{u}}_i^{(n)} = u_i^{(n)} - \frac{\Delta t}{\Delta x}\left(\overline{f}_i - \overline{f}_{i-1}\right) \qquad \text{Corrector} \tag{6.151b}$$

$$u_i^{(n+1)} = \frac{1}{2}\left(\overline{u}_i + \overline{\overline{u}}_i\right) \tag{6.151c}$$

which suggests the alternative method

$$\overline{u}_i = u_i^{(n)} - \frac{\Delta t}{\Delta x}\left(f_i^{(n)} - f_{i-1}^{(n)}\right) \qquad \text{Predictor} \tag{6.152a}$$

$$\overline{\overline{u}}_i^{(n)} = u_i^{(n)} - \frac{\Delta t}{\Delta x}\left(\overline{f}_{i+1} - \overline{f}_i\right) \qquad \text{Corrector} \tag{6.152b}$$

$$u_i^{(n+1)} = \frac{1}{2}\left(\overline{u}_i + \overline{\overline{u}}_i\right) \tag{6.152c}$$

where the order of the forward and backward differences is reversed. For linear problems, the switch in order has no effect but for nonlinear flux functions, the results will be different between the two methods. Generally speaking, the first method works better for left-to-right shocks while the second methods is preferable for right-to-left shocks.

Figure 6.5: Example mesh for conservative discretization

6.9.5 Conservative Discretizations

Integrating equation (6.136) over a domain $x \in [A, B]$ yields

$$\int_{x_A}^{x_B} \frac{\partial u}{\partial t} dx + \int_{x_A}^{x_B} \frac{\partial f(u)}{\partial x} dx = 0 \tag{6.153}$$

since the second term is a perfect integral, we obtain

$$\int_{x_A}^{x_B} \frac{\partial u}{\partial t} dx + f(u)|_{x_B} - f(u)|_{x_A} = 0 \tag{6.154}$$

If instead, we had divided the original domain from A to B into three subdomain, we could have performed the same integration leading to (6.154) by integrating over each subdomain and summing the results from each subdomain. Since all interior fluxes cancel, we are left with the same result as if we had originally integrated on a single large domain.

This conservation property is automatically satisfied in the continuous case, but we have to be very careful to make sure that it is retained after discretization. For example, consider the mesh shown in figure 6.5. Using a second-order central discretization in space leads to the semi-discrete equation at node i of the form

$$\frac{\partial u_i}{\partial t} + \frac{1}{\Delta x}\left(f_{i+1/2} - f_{i-1/2}\right) = 0 \tag{6.155}$$

with similar equations at nodes $i+1$ and $i-1$. If we sum the discrete equations at $i-1$, i, and $i+1$ then we obtain

$$\frac{1}{3}\frac{\partial}{\partial t}(u_{i-1} + u_i + u_{i+1}) + \frac{1}{3\Delta x}\left(f_{i+3/2} - f_{i-3/2}\right) = 0 \tag{6.156}$$

which is a consistent discretization of the original equation on the larger domain, $A - B$ in figure 6.5. The essential requirement for this to be true is that all interior fluxes cancel, just like in the continuous case, leaving only the fluxes evaluated at the outer boundaries. A numerical method which has this property guarantees global conservation.

In general, a conservative discretization of the scalar model problem (6.136) can be written in the form

$$u_i^{(n+1)} = u_i^{(n)} - \frac{\Delta t}{\Delta x}\left(f^*_{i+1/2} - f^*_{i-1/2}\right) \tag{6.157}$$

where $f*_i$ is the *numerical flux* for a given method. If the method can be written in this form, then it is discretely conservative.

As an example, MacCormack's method is given by the numerical flux

$$f^*_{i+1/2} = \frac{f^{(n)}_{i+1} + \overline{f}_i}{2} \tag{6.158}$$

which demonstrates that MacCormack is a conservative numerical method.

Similarly, the numerical flux for the Lax-Wendroff method is

$$f^*_{i+1/2} = \frac{1}{2}\left(f^{(n)}_{i+1} + f^{(n)}_i\right) - \frac{\Delta t}{\Delta x} a^{(n)}_{i+1/2}(f^{(n)}_{i+1} - f^{(n)}_i) \tag{6.159}$$

which indicates that this is also a conservative numerical method, although the numerical flux is quite a bit for complex than that for MacCormack.

6.9.6 Nonconservative Discretizations

To see the effect of a nonconservative numerical method, let's start with the nonconservative form of the model problem

$$\frac{\partial u}{\partial t} + a(u)\frac{\partial u}{\partial x} = 0 \tag{6.160}$$

and use a simple second-order discretization of the form

$$\frac{\partial u_i}{\partial t} + \frac{a_i}{\Delta x}\left(u_{i+1/2} - u_{i-1/2}\right) = 0 \tag{6.161}$$

where $a_i = (a_{i+1/2} + a_{i-1/2})/2$ and we can write similar equations at nodes $i+1$ and $i-1$. Summing the discrete equations at $i-1$, i, and $i+1$ leads to

$$\begin{aligned} \frac{1}{3}\frac{\partial}{\partial t}(u_{i-1} + u_i + u_{i+1}) + \frac{(a_{i+3/2} + a_{i-3/2})}{2}\frac{1}{3\Delta x}\left(u_{i+3/2} - u_{i-3/2}\right) &= \\ (a_{i+3/2} - a_{i-1/2})\frac{(u_{i+1/2} - u_{i-3/2})}{6\Delta x} - (a_{i+1/2} - a_{i-3/2})\frac{(u_{i+3/2} - u_{i-1/2})}{6\Delta x} &= \\ \Delta x^2\left[(a_{,x}u_{,x})_{,x} - (a_{,x}u_{,xx})\right] \end{aligned} \tag{6.162}$$

which is not consistent with a discretization on the larger interval from A to B. The difference is due to the numerical source term which result from the incomplete cancelation of interior numerical fluxes. Although (6.161) is nonconservative, it is of course still a second-order accurate discretization of the PDE. However, the errors appear here as numerical sources of quantity u

which are not present in the continuous equation or in the conservative discretization. The result is that if you use a nonconservative equation for a problem in which mass is supposed to be a conserved quantity, unlike a conservative discretization, the nonconservative numerical solution will not hold a constant mass. In some cases this can lead to significant errors, especially when there are regions of marginally or under-resolved gradients. The classic example of this in CFD is shock capturing where the grid is intentionally too coarse to represent the sharp gradients of a shock wave. Under these conditions, a conservative method will still give the correct jump in quantities across the shock (the Rankine–Hugoniot relations) while a nonconservative formulation will not.

6.10 Example: Inviscid Burgers Equation

As an example of the various methods for hyperbolic equations that we have disussed, consider the inviscid Burgers equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0. \tag{6.163}$$

By defining the flux function $f(u) = u^2/2$ then we can rewrite Burgers equation in the conservation form

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} = 0. \tag{6.164}$$

This equation is solved on the domain $x \in [0, 4]$ subject to the boundary condition and initial condition

$$u(0,t) = 1 \qquad u(x,0) = \begin{cases} 1 & 0 \le x \le 2 \\ 0 & 2 < x \le 4 \end{cases} \tag{6.165}$$

This problem is solved using the Lax-Friedrich, Richtmyer, and MacCormack methods and the solutions are shown in figure 6.6 all for $CFL = 1$ and 101 node points in x.

Remarks:

1. The Lax-Friedrichs solution is too dissipative with the shock spread over and increasing number of grid points as time increases. This is inaccuracy is not surprising since the method is only first-order in time.

2. The Richtmyer solution is plaqued by dispersion errors which lead to oscillations (mainly overshoots) in the solution near the shock.

3. Of the three methods, MacCormack gives the best solutions with the shock spread consistently over four cells with no oscillations. In fact, it can be proved that the MacCormack method provides the optimal dissipation for this problem.

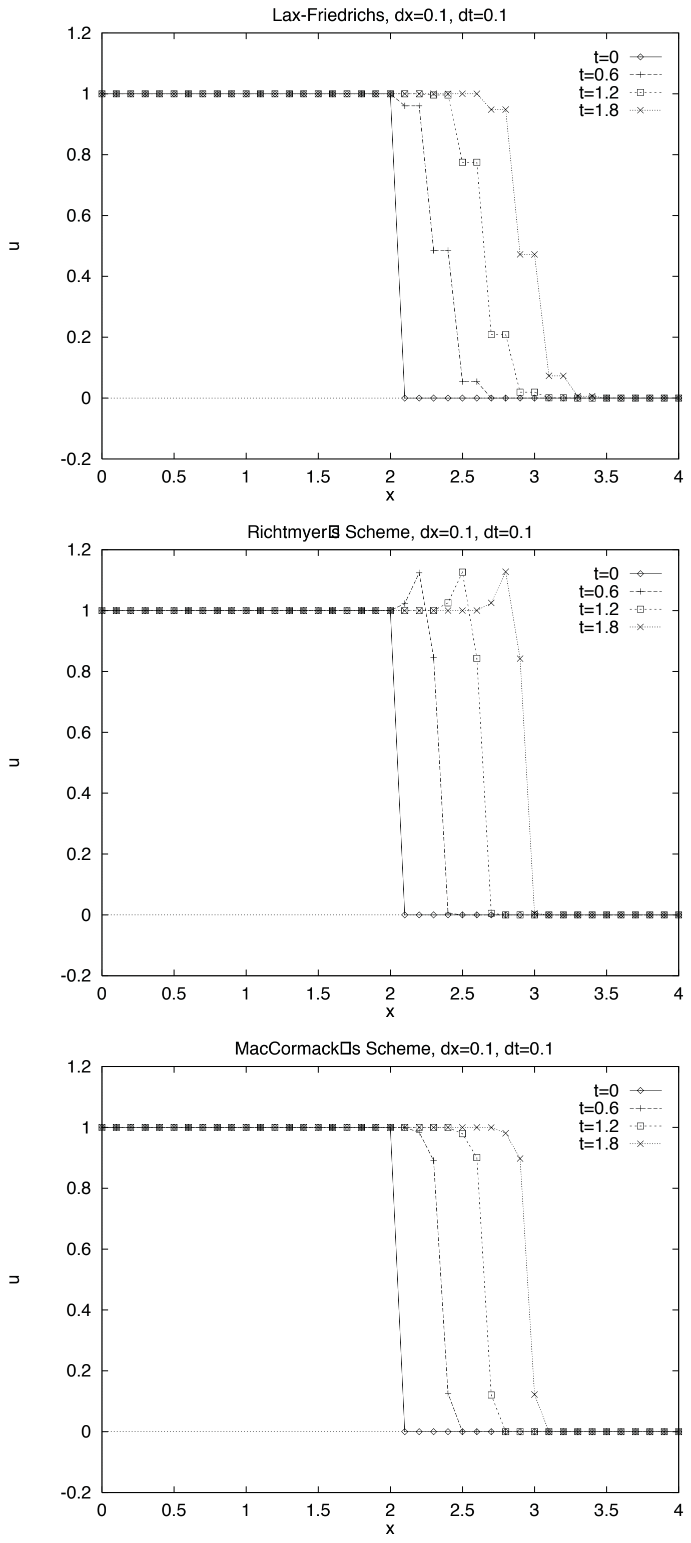

Figure 6.6: Propagating shock problem solved using three numerical methods: Lax-Friedrich, Richtmyer, and MacCormack (from top to bottom) with $N_x = 101$ and $CFL = 1$.

Chapter 7

Numerical Solution of the Navier–Stokes Equations

Note: This chapter is preliminary, but is included in this form in the hope that it contains some information of use to the reader.

7.1 Navier–Stokes

The incompressible Navier–Stokes equations for velocity vector $\boldsymbol{u}$ and pressure p are given by

$$\rho\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\boldsymbol{\nabla})\boldsymbol{u} + \boldsymbol{\nabla} p = \mu\nabla^2\boldsymbol{u} \tag{7.1a}$$

$$\boldsymbol{\nabla}\cdot\boldsymbol{u} = 0 \tag{7.1b}$$

where ρ is the fluid density (mass per unit volume) and mu is the coefficient of viscosity. The first equation represents conservation of linear momentum while the second equation enforces conservation of mass or continuity. This form of the Navier–Stokes equations is very general since the vector notation makes it independent of coordinate system and the number of spatial dimensions. Often we will find it convenient to work in Cartesian coordiates were we can rewrite the Navier–Stokes equations using the concise form

$$\rho u_{i,t} + \rho u_j u_{i,j} + p_{,i} = \mu u_{i,jj} \tag{7.2a}$$

$$u_{i,i} = 0 \tag{7.2b}$$

where subscripts preceeding a comma indicate the spatial coordinate and subscripts after a comma denote differentiation. As a general rule, repeated indices imply summation over all the possible combinations (the Einstein summation convention). Thus, in three spatial-dimensions the velocity vector is given by $\boldsymbol{u} = \{u_1, u_2, u_3\}^T = \{u, v, w\}^T$,

$$u_{i,jj} \equiv u_{i,11} + u_{i,22} + u_{i,33}, \tag{7.3}$$

and

$$u_j u_{i,j} \equiv u_1 u_{i,1} + u_2 u_{i,2} + u_3 u_{i,3}\ . \tag{7.4}$$

The form of the Navier-Stokes momentum equation given above (7.2a) is in a nonconservative form. Using the continuity equation, we can easily varify that the conservation form of the momentum equation is

$$(\rho u_i)_{,t} + (\rho u_j u_i)_{,j} + p_{,i} = (\mu u_{i,j})_{,j} \tag{7.5}$$

where each term can now be interpretted as the flux of a quantity in either space or time. Note that in constructing this equation from (7.2a) we must assume that ρ and mu are constants in space and/or time. In general, (7.5) is actually the correct form which allows for variable fluid properties. Here we will limite ourselves to constant properties so that by introducing a suitable set of reference scales, we can write the equations in nondimensional form as

$$u_{i,t} + (u_j u_i)_{,j} + p_{,i} = \frac{1}{\mathit{Re}} u_{i,jj} \tag{7.6a}$$

$$u_{i,i} = 0 \tag{7.6b}$$

where $\mathit{Re} = \rho U L/\mu$ is the reference Reynolds number, U is a reference velocity, and L is a reference length.

Now we are ready to consider discretizations of the Navier–Stokes system. However, we are first met immediately by a harsh reality: there is no dynamic equation for pressure, and likewise, the continuity equation has no time derivative. This is in fact the primary difficulty of CFD for incompressible flows and there are a variety of ways to overcome this impediment documented in the liturature. Instead of reviewing all the options, we will take one particular solutions and discuss it in detail

7.2 The Fractional Step Method

One way to think of the continuity equation is that it imposes a constraint on the velocity field that such that the velocity must always be a divergence-free vector field. Under this interpretation, the pressure then plays the role of a Lagrange multiplier that is used to enforce the divergence free constraint. To derive an explicit equation for the pressure, we can take the divergence of the momenutum equations (7.5)$_{,i}$ which yields

$$p_{,ii} = -(u_j u_i)_{,ij} \tag{7.7}$$

which we recognize as simply a Poisson equation for the pressure. Thus, one approach is to discretize the momentum equation and solving an appropriate Poisson equation for the pressure to ensure that the velocity field is divergence-free. This is the so-called fractional-step method.

To keep things simple, let's first a semi-discrete method using explicit Euler in time. The Navier–Stokes equations can then be written in the form

$$\begin{bmatrix} \boldsymbol{I} & \Delta t \boldsymbol{\nabla} \\ \boldsymbol{\nabla}\cdot & 0 \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}^{(n+1)} \\ p^{(n+1)} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{r}^{(n)} \\ 0 \end{Bmatrix} \tag{7.8}$$

where

$$\boldsymbol{r}^{(n)} = \boldsymbol{u}^{(n)} - \Delta t \left[(\boldsymbol{u}\cdot\boldsymbol{\nabla})\boldsymbol{u} - \frac{1}{Re}\nabla^2 \boldsymbol{u} \right]^{(n)} \tag{7.9}$$

and we have used the vector notation for brevity. This equation can be factored into

$$\begin{bmatrix} \boldsymbol{I} & 0 \\ \boldsymbol{\nabla}\cdot & -\Delta t \boldsymbol{\nabla}\cdot\boldsymbol{\nabla} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \Delta t \boldsymbol{\nabla} \\ 0 & \boldsymbol{I} \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}^{(n+1)} \\ p^{(n+1)} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{r}^{(n)} \\ 0 \end{Bmatrix} \tag{7.10}$$

which can be decomposed into two systems:

$$\begin{bmatrix} \boldsymbol{I} & 0 \\ \boldsymbol{\nabla}\cdot & -\Delta t \boldsymbol{\nabla}\cdot\boldsymbol{\nabla} \end{bmatrix} \begin{Bmatrix} \hat{\boldsymbol{u}}^{(n+1)} \\ \hat{p}^{(n+1)} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{r}^{(n)} \\ 0 \end{Bmatrix} \tag{7.11a}$$

$$\begin{bmatrix} \boldsymbol{I} & \Delta t \boldsymbol{\nabla} \\ 0 & \boldsymbol{I} \end{bmatrix} \begin{Bmatrix} \boldsymbol{u}^{(n+1)} \\ p^{(n+1)} \end{Bmatrix} = \begin{Bmatrix} \hat{\boldsymbol{u}}^{(n+1)} \\ \hat{p}^{(n+1)} \end{Bmatrix} \tag{7.11b}$$

This leads to the method:

$$\hat{\boldsymbol{u}}^{(n+1)} = \boldsymbol{r}^{(n)} \tag{7.12a}$$

$$\nabla^2 \hat{p}^{(n+1)} = \boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\hat{p}^{(n+1)} = \frac{1}{\Delta t}\boldsymbol{\nabla}\cdot\hat{\boldsymbol{u}}^{(n+1)} \tag{7.12b}$$

$$\boldsymbol{u}^{(n+1)} = \hat{\boldsymbol{u}}^{(n+1)} - \Delta t \boldsymbol{\nabla}\hat{p}^{(n+1)} \tag{7.12c}$$

This is the basic idea behind the fractional step method. In eccense, the momentum equation is solved without the pressure gradient giving an intermediate velocity, $\hat{\boldsymbol{u}}$. Then a Poisson equation is solved for the pressure which is used to project the intermediate velocity to be divergence-free.

A slight, but important modification of this method is to rewrite the equation in the so-called delta-form

$$\begin{bmatrix} \boldsymbol{I} & \Delta t \boldsymbol{\nabla} \\ \boldsymbol{\nabla}\cdot & 0 \end{bmatrix} \begin{Bmatrix} \delta\boldsymbol{u}^{(n+1)} \\ \delta p^{(n+1)} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{r}^{(n)} - \boldsymbol{u}^{(n)} - \Delta t \boldsymbol{\nabla} p^{(n)} \\ 0 \end{Bmatrix} \tag{7.13}$$

which leads to the method:

$$\delta\hat{\boldsymbol{u}}^{(n+1)} = \boldsymbol{r}^{(n)} - \boldsymbol{u}^{(n)} - \Delta t \boldsymbol{\nabla} p^{(n)} \tag{7.14a}$$

$$\nabla^2 \delta\hat{p}^{(n+1)} = \boldsymbol{\nabla}\cdot\boldsymbol{\nabla}\delta\hat{p}^{(n+1)} = \frac{1}{\Delta t}\boldsymbol{\nabla}\cdot\delta\hat{\boldsymbol{u}}^{(n+1)} \tag{7.14b}$$

$$\boldsymbol{u}^{(n+1)} = \boldsymbol{u}^{(n)} + \delta\hat{\boldsymbol{u}}^{(n+1)} - \Delta t \boldsymbol{\nabla}\delta\hat{p}^{(n+1)} \tag{7.14c}$$

The effect of working in the delta form is that we can see from (7.14c) that the $\hat{\boldsymbol{u}}$ velocity is a second-order approximation to the velocity $\boldsymbol{u}^{(n+1)}$ so that we can use the same boundary conditions on the $\hat{\boldsymbol{u}}$ as used for $\boldsymbol{u}$.

In the spatially continuous case (which is what we have done so far) everything is fine. However, when we introduce finite-difference approximations for the spatial derivatives, we may introduce an instability if we do not handle the pressure carefully. This instability results from an overspecification of the pressure leading to checkboard type oscillations in the pressure that are unphysical. The classical approach to correcting this is to use fewer pressure nodes compared to velocity nodes. This approach often takes the form of a staggered mesh where the pressure nodes are at cell centers and the velocity nodes are at cell faces.

7.3 Upwinding and Numerical dissipation

In the early day of computational fluid dynamics, it was recognized that Galerkin and other “central” difference type methods can have difficulties for fluid flows that are advection dominated. Since most of the early work in CFD was based on finite difference methods, we take a small diversion and review the subject of upwind finite difference methods which have been developed to address the limitations of central difference methods. We then, review the more recent advances in “upwind” or stabilized methods for finite element discretizations such as SUPG and GLS. Our motivation for exploring upwind and stabilized methods is that we have the goal of applying the techniques of optimal control, discussed in previous chapters for the linear advection-diffusion equations, to problems in fluid mechanics where upwind type methods are commonly used. Specifically, we want to explore the impact of these numerical methods for computing adjoint quantities required to solve optimal control problems.

7.4 Finite Difference Methods

7.4.1 Model problem

Consider the simple, steady, one-dimensional advection diffusion equation

$$\begin{aligned} ay'(x) - \nu y''(x) &= 0 \quad \text{in } (0,1), \\ y(0) &= 0, \\ y(1) &= 1 \end{aligned}$$

where a is the convection velocity and ν is the diffusion (or viscosity) coefficient. This equation with boundary conditions has the exact solution

$$y(x) = \frac{\exp\left(\frac{ax}{\nu}\right) - 1}{\exp\left(\frac{a}{\nu}\right) - 1}.$$

and the solution has a boundary layer located at $x = 1$ for large values of the ratio a/ν.

7.4.2 Central Difference

Since the initial research in upwind methods was in the context of finite difference, we briefly consider a finite difference solution to equation (7.15). First, let's try a central difference approximation

$$a\frac{y_{i+1}^h - y_{i-1}^h}{2h} - \nu\frac{y_{i+1}^h - 2y_i^h + y_{i-1}^h}{h^2} = 0,$$

where y_i^h denotes our numerical solution at node point i. Multiplying through by $2h/a$ gives

$$y_{i+1}^h - y_{i-1}^h - \frac{\nu 2}{ah}\left(y_{i+1}^h - 2y_i^h + y_{i-1}^h\right) = 0,$$

And we recongnize the parameter $\alpha = ah/2\nu$ as a Reynolds number, often called the element or local Reynolds number since the reference length is h, the local element size. If $\alpha > 1$ then locally the flow is convection dominated for the mesh under consideration and typically central difference type methods lead to solutions with numerical oscillations. For the first example problem, where you had $\nu = 0.01$ and $a = 1$, you would observe oscillations for $h > 0.02$ which means that you would need a mesh with $n_e > 50$ to get a smooth solution with a uniform mesh. Figure 7.4.2

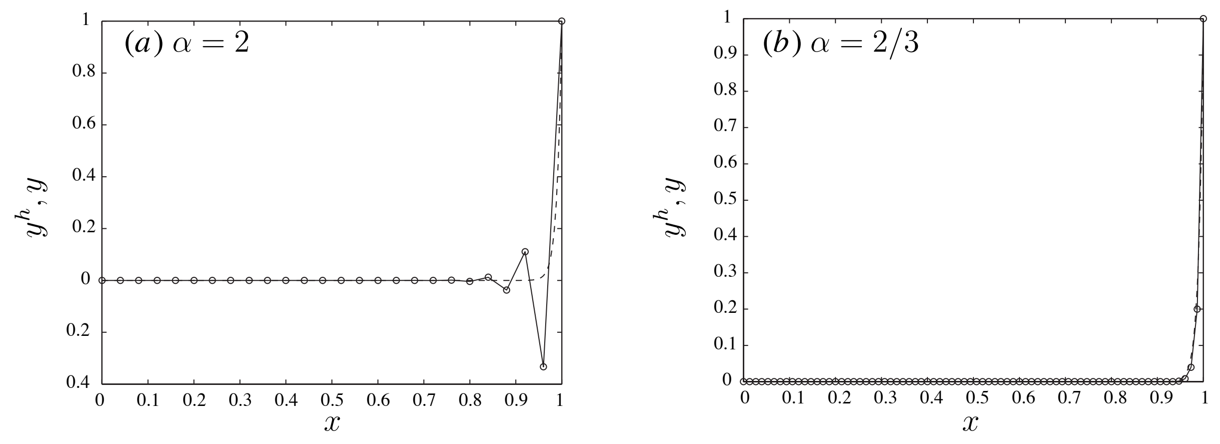

Figure 7.1: Numerical solutions to (7.15) using second order central differences (—o—) compared to the exact solution (----): (a) $\alpha = 2$, $n_e = 25$; (b) $\alpha = 2/3$, $n_e = 75$.

shows the effect of element Reynolds number on the quality of the numerical solution to (7.15) using central differences – for $\alpha = 2$ the numerical solution has large, nonphysical oscillations, while for $\alpha = 2/3$ the solution is smooth. From this figure it appears that the central difference approximation is *underdiffuse*. Using a uniform mesh that gives $\alpha \leq 1$ is prohibitively expensive, since the only region where high resolution is required is in the boundary layer. The only hope for smooth, *efficient* solutions is to design a nonuniform mesh which ensures that locally the element Reynolds number is less than one and this was exactly what you did in the first homework set. The bottomline is that central difference methods can place rather severe resolution requirements for convection dominated flows and this has lead to the development of upwind type methods in an attempt to circumvent this.

7.4.3 Upwind Difference

Consider the upwind difference of the same advection diffusion equation where we assume that $a > 0$ so that

$$a\frac{y_i^h - y_{i-1}^h}{h} - \nu\frac{y_{i+1}^h - 2y_i^h + y_{i-1}^h}{h^2} = 0.$$

Notice that we have applied a backward difference formula for the convection term that is "upwind" for positive a, while we have retained the central difference approximation to the second derivative. By using an upwind difference on the advection term, we are recognizing that the advection term moves information from left to right for positive a. Thus, we use a finite difference formula that is biased towards the left.

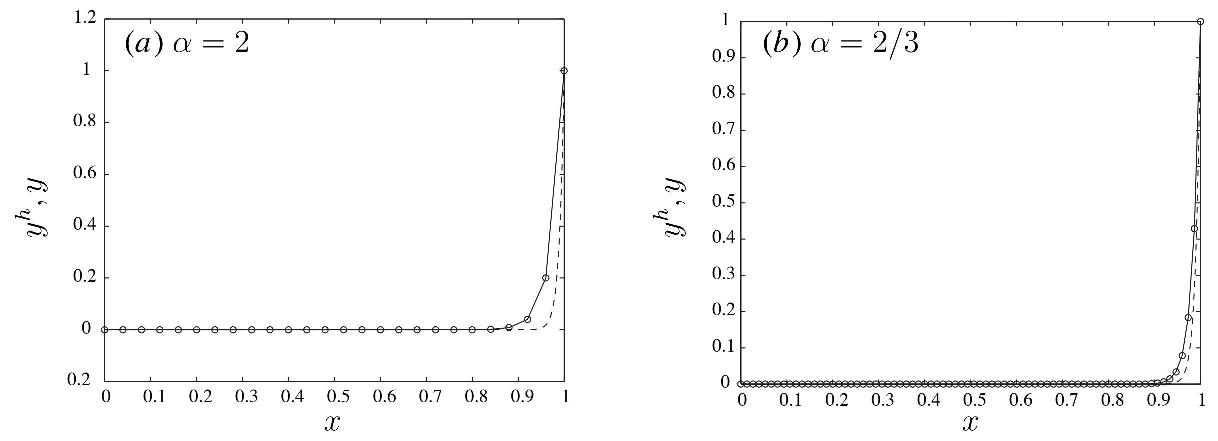

Figure 7.2: Numerical solutions to (7.15) using first-order upwind differences (—o—) compared to the exact solution (----): (a) $\alpha = 2$, $n_e = 25$; (b) $\alpha = 2/3$, $n_e = 75$.

Although this method is only first order accurate, the introduction of the upwind difference will provide oscillation free solutions as shown in figure 7.4.3, even for large values of α. Unfortunately, we see that due to the first-order accuracy of the upwind first-derivative approximation, the accuracy of the solution is not very good. In fact, even for small α the method is *overdiffuse*. One way of correcting this is to use higher-order one-sided and biased difference formulae. In general, these will have less inherent numerical dissipation than the first-order methods discussed here so that you could tune the difference formula to provide an appropriate amount of dissipation. Unfortunately, this may require a different finite difference formula for each problem solved.

Another approach to reducing the inherent dissipation in the first-order upwind scheme is to rewrite the upwind formulation in the following way

$$a\frac{y^h_{i+1} - y^h_{i-1}}{2h} - \left(\nu + \frac{h|a|}{2}\right)\frac{y^h_{i+1} - 2y^h_i + y^h_{i-1}}{h^2} = 0 \qquad (7.15)$$

which we see is equivalent to using central differences for the advection term but with an added numerical dissipation coefficient of $h|a|/2$ that depends on the mesh spacing.[1] Of course, as $h \to 0$ the numerical dissipation approaches zero, but only at first-order. Since we showed that the amount of diffusion naturally present in the upwind scheme is too much while that in a pure central scheme is underdiffuse, we could consider introducing a numerical viscosity parameter, $\tilde{\xi}$, which we tune

[1]I have used $|a|$ here so that this method works for both positive and negative a.

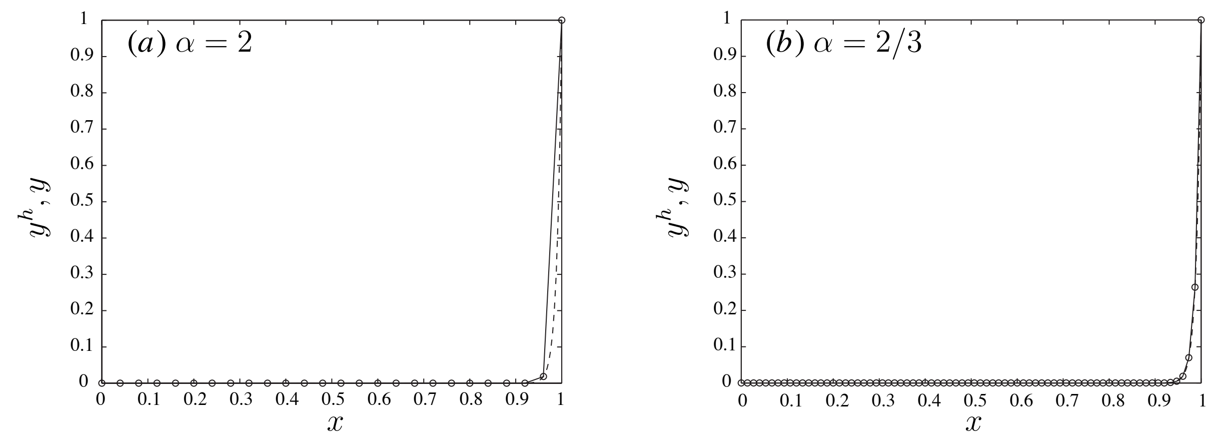

Figure 7.3: Numerical solutions to (7.15) using tunned second-order dissipation (—○—) compared to the exact solution (----): (a) $\alpha = 2$, $n_e = 25$; (b) $\alpha = 2/3$, $n_e = 75$.

to give the best solution. The method would then be,

$$a\frac{y^h_{i+1} - y^h_{i-1}}{2h} - \left(\nu + \tilde{\xi}\frac{h|a|}{2}\right)\frac{y^h_{i+1} - 2y^h_i + y^h_{i-1}}{h^2} = 0 \tag{7.16}$$

and it turns out that for a given α, one can select a value of $\tilde{\xi}(\alpha) = \coth(\alpha) - 1/\alpha$ such that the solution is nodally exact as shown in figure 7.4.3 [1]. We have apparently achieved the holy grail of numerical analysis. The catch, or course, is that while we may be able to construct $\tilde{\xi}(\alpha)$ for this simple one-dimensional linear model problem, in more realistic situations this is not generally possible. In particular, in multidimensions, upwind solutions often exhibit excessive dissipation normal to the flow direction. Likewise, problems can also occur in transient problems and equations with strong source terms. However, we can use the experience gleaned from upwind methods as guidence in designing numerical methods for more complex problems.

7.4.4 Interpretation in terms of Iterative Methods

We can also view the influence of the advection term from a linear algebra point of view by accessing the impact of discretization on the spectral radius of standard iterative methods for solving linear systems of equations. All of the methods discussed so far can be written in the form

$$\boldsymbol{M}\boldsymbol{y}^h = \boldsymbol{r} \tag{7.17}$$

where $\boldsymbol{y}^h = \{y^h_1, y^h_2, \ldots, y^h_n\}^T$ with n equal to the number of node points. In this expression, $\boldsymbol{M}$ is a $n \times n$ matrix and $\boldsymbol{r} = \{r_1, r_2, \ldots, r_n\}^T$ is composed of all terms not explicitly dependent on

$\boldsymbol{y}^h$ including boundary conditions. For all the discretizations presented so far, the matrix $\boldsymbol{M}$ has a tridiagonal structure. It is instructive to examine the structure of the different matrices arising from each method. For the central difference method

$$\boldsymbol{M}_{central} = \frac{a}{2h}\boldsymbol{B}[-1,0,1] - \frac{\nu}{(h)^2}\boldsymbol{B}[1,-2,1] \tag{7.18}$$

where $\boldsymbol{B}[a,b,c]$ denotes a tridiagonal matrix with lower diagonal a, diagonal b, and upper diagonal c. For the first-order upwind method

$$\boldsymbol{M}_{upwind} = \frac{a}{h}\boldsymbol{B}[-1,1,0] - \frac{\nu}{(h)^2}\boldsymbol{B}[1,-2,1] \tag{7.19}$$

and for the tuned dissipation method

$$\boldsymbol{M}_{tuned} = \frac{a}{2h}\boldsymbol{B}[-1,0,1] - \frac{\nu + \tilde{\xi}\frac{h|a|}{2}}{(h)^2}\boldsymbol{B}[1,-2,1]\,. \tag{7.20}$$

From these expressions, we can draw the following general conclusions:

1. Central difference discretization of the diffusion term leads to a symmetric, positive definite, contribution to the system matrix – all the eigenvalues of this matrix are real and greater than zero (for $\nu > 0$).

2. In contrast, a central difference approximation to the advection term results in a skew-symmetric contribution to $\boldsymbol{M}$. The eigenvalues of this contribution are all pure imaginary.

3. For large α, the skew-symmetric component for the central scheme dominates the symmetric component of the matrix $\boldsymbol{M}$. This results in a matrix that is *not* diagonally dominate. In fact, for $\alpha > 1$, iterative methods like point Jacobi, and Guass–Seidel will not even converge.

4. For the upwind method, we see that the advection contribution is lower triangular, with a positive diagonal. Thus, for all α the advection term increases the diagonal dominance of $\boldsymbol{M}$ resulting in unconditional convergence for point Jacobi and Gauss–Seidel.

5. Similarly, we see that for the tuned dissipation approach, the added term increases the diagonal dominance of the matrix $\boldsymbol{M}$ as long as $\tilde{\xi} > 0$. If the optimal value of $\tilde{\xi}$ is used then point Jacobi and Gauss–Seidel will converge for all α.

Thus, the addition of numerical dissipation through either upwind differences or an explicit dissipation term leads to matrices that are better suited to iterative solution techniques. This turns

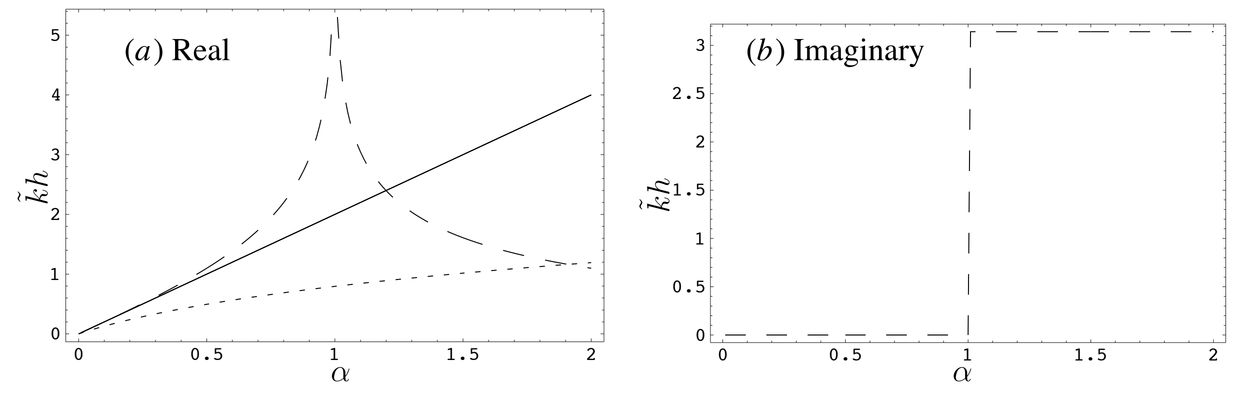

Figure 7.4: Comparison of the real (a) and imaginary (b) components of $\tilde{k}h$ as a function of α for the exact and optimally tuned solution ——, the central difference scheme ----, and the upwind method ········ .

out to be one of the major advantages of upwind and stabilized schemes when applied to very large advection dominated problems.

7.5 Theoretical perspective

The analytical solutions to the steady advection-diffusion equation with constant coefficients are of the form

$$y(x) = \tilde{y}_0 + \tilde{y}_1 e^{ax/\nu} \tag{7.21}$$

where the two unknown coefficients are determined by the boundary conditions. Thus, without loss of generality we can consider fundamental solutions of the form

$$y(x) = \tilde{y}e^{kx} \tag{7.22}$$

Plugging this expession into the exact equation leads to

$$kh - \frac{1}{\alpha}\frac{(kh)^2}{2} = 0. \tag{7.23}$$

where we have introduced the mesh size h for convenience in comparision with the discrete results below. The exact solutions to this characteristic equation are; $kh = 0$ and $kh = 2\alpha$ which is in agreement with (7.21).

Similarly, the discrete solutions are assumed to be of the form

$$y_i = \tilde{y}e^{\tilde{k}x_i} \tag{7.24}$$

which, when substituted into the central difference approximation leads to

$$\sinh(\tilde{k}h) + \frac{1}{\alpha}[1 - \cosh(\tilde{k}h)] = 0 \tag{7.25}$$

Solving this approximate characteristic equation leads to

$$\tilde{k}h = 0, \qquad \tilde{k}h = \cosh^{-1}\left(\frac{1+\alpha^2}{1-\alpha^2}\right) \tag{7.26}$$

Figure 7.5 shows a comparison of the nontrivial roots as a function of α. As expected, for small α the central difference method is in good agreement with the exact solution. However, as α approaches one from below, the central difference scheme overpredictes the exact growth factor indicated that the method is underdiffuse. For $\alpha > 1$ the central difference $\tilde{k}h$ becomes complex with the imaginary component always $\Im(\tilde{k}h) = \pi$. This means that the numerical solution will have oscillations with wavelength equal to $2h$ for all $\alpha > 1$. These are the so-called node-to-node oscillations or 2δ waves commonly seen in underresolved numerical solutions using central differencing. The real part of $\tilde{k}h$ actually indicates that the solution is overdiffuse for $\alpha > 1.25$ which accounts for the fact that the oscillations tend to spread out for large α.

If instead of central difference, we used the upwind method, then we would get the characteristic equation

$$\sinh(\tilde{k}h) + \left(\frac{1}{\alpha} + 1\right)[1 - \cosh(\tilde{k}h)] = 0 \tag{7.27}$$

which yields

$$\tilde{k}h = 0 \qquad \tilde{k}h = \cosh^{-1}\left(\frac{1+2\alpha+2\alpha^2}{1+2\alpha}\right) \tag{7.28}$$

while for the optimally tuned dissipation we get

$$\sinh(\tilde{k}h) + \coth(\alpha)[1 - \cosh(\tilde{k}h)] = 0 \tag{7.29}$$

which yields $\tilde{k}h = 0$ and $\tilde{k}h = 2\alpha$ in agreement with the exact solution. The nontrivial $\tilde{k}h$ are plotted in figure 7.5 for all methods. Clearly, the upwind method is overdiffuse for all α, but the imaginary part is zero indicating that solutions are oscillation free.

So far we have focused on the advection-diffusion model problem. It is also instructive to look at the resolving characteristics of individual finite difference approximations. Assume that we have a uniform mesh where the function y satisfies periodic boundary conditions such that

$$y(x) = \tilde{y}e^{ikx} \tag{7.30}$$

where k is the wavenumber and $i = \sqrt{-1}$. Note that in the general case we would have a summation of complex exponentials over a range of wavenumbers, but it is sufficient to consider a single mode of this summation here.

With this form, the first derivative is given by

$$\frac{dy}{dx} = ik\tilde{y}e^{ikx} \tag{7.31}$$

In practice, we must approximate the derivatives using, say, a finite difference formula. If we use the second order central difference scheme then

$$\frac{dy_i}{dx} = \frac{y_{i+1} - y_{i-1}}{2h} \tag{7.32}$$

if we assume that the discrete function can be written in the form

$$y_i = \tilde{y}e^{ikx_i} \tag{7.33}$$

then we get

$$\frac{dy_i}{dx} = \frac{\tilde{y}}{2h}\left(e^{ikx_{i+1}} - e^{ikx_{i-1}}\right) = i\frac{\sin(kh)}{h}\tilde{y} = i\tilde{k}\tilde{y} \tag{7.34}$$

Thus, we see that the finite difference approximation leads to the same form of the derivative except with a *modified wave number*, $\tilde{k} = \sin(kh)/h$ instead of the real wave number, k. Figure 7.5 shows a plot of the modified wave number for the central difference scheme. As you can see, the finite difference scheme only does a reasonable job of approximating the real wavenumber for $kh < 1$ which cooresponds to n_e/wave $> 2\pi$. Thus you must have more that 6 cells per wavelength in the solution to get accurate results. One can see that the errors in the finite difference scheme for $kh > 1$ are such that the modified wavenumber is less than the exact wavenumber. This means that for unsteady problems (like the wave equation) the high wavenumber components of the numerical solution will travel with a phase speed that is too low – the error is dispersive. In fact, the mode with $kh = \pi$ will have *zero* phase speed, while this is the fastest moving component of the exact solution. However, since the wavenumbers are all real, there are no dissipative errors in the central difference approximation. That is why you will sometimes hear people say that central difference methods have no numerical diffusion. However, the disspersion errors manifest themselves as oscilations in the solution as energy is distributed incorrectly in space.

The same approach can be taken for the second derivative, and the standard central difference

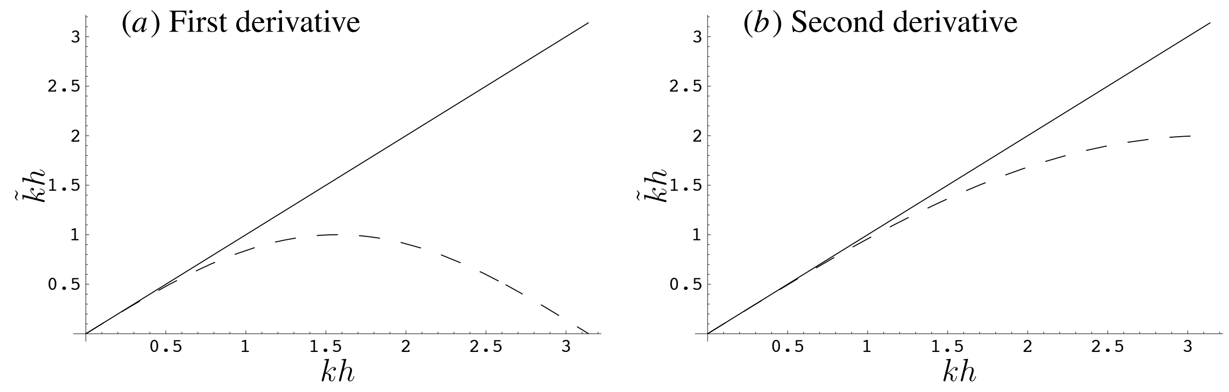

Figure 7.5: Comparison of the first (a) and second (b) derivative approxitions in terms of modified wave number: exact solution —— and the central difference scheme - - - -.

approximation leads to a modified wave number given by

$$\tilde{k}h = \sqrt{2[1 - \cos(kh)]} \tag{7.35}$$

Figure 7.5 also shows this modified wavenumber again compared to the exact wavenumber. For reasonable accuracy we require $kh < 1.5$ which is a little better than that for the first derivative. Similar to the first derivative, the modifed wavenumber for the second derivative is less than the exact wavenumber for large kh. Thus, the diffusion of the large wavenumber is *underpredicted*! This, in fact, is the reason that central approxiations to the advection-diffusion equation have oscillations on coarse grids – the numerical solution tends to be underdiffuse in the high wavenumbers.

We now return to the upwind difference approximation of the first derivative, realizing that it can be written as a combination of central difference approximations to the first and second derivatives

$$\tilde{k}h = \sin(kh) + i(\cos(kh) - 1) \tag{7.36}$$

The real part is identical to the central difference, but now we have an imaginary component of the wavenumber that is dissapative. Figure 7.5 shows the real and imaginary components as a function of kh. For the high wavenumbers that have large errors in $\Re(\tilde{k}h)$ the numerical dissipation is also large so that these wavenumbers tend to be smoothed out.

7.5.1 Summary

In summary, the use of upwind methods was motivated by the desire to obtain accurate solutions to advection dominated flows without having to resort to the excessive resolution required to get smooth numerical solutions with standard central difference methods. Although upwind methods

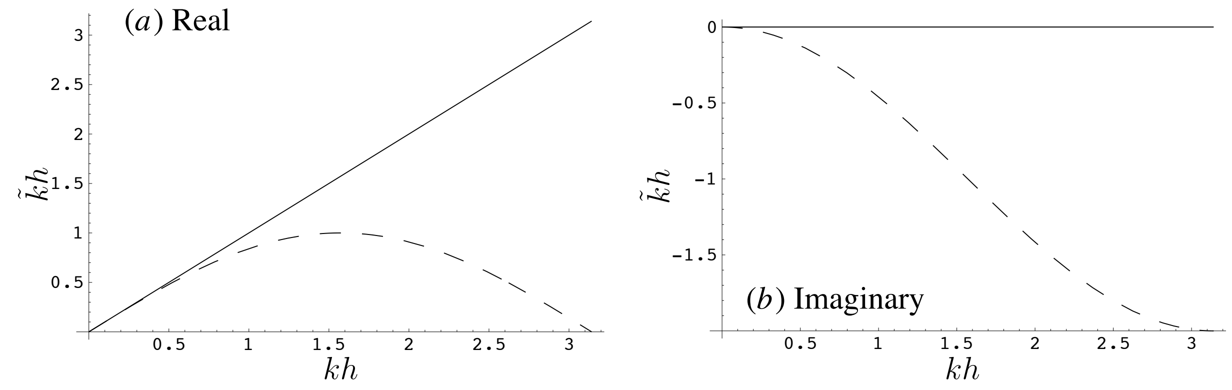

Figure 7.6: Comparison of the real (a) and imaginary(b) components of the upwind first derivative approximation in terms of modified wave number: exact solution —— and the upwind difference scheme - - - - .

have the desirable characteristic that numerical dissipation is "automatically" included, the numerical analyst doesn't have direct control over the amount of dissipation added. However, by writing an upwind method as an equivalent central difference method plus numerical dissipation, we can identify and control the amount of dissipation added to the numerical solution – in some cases even recovering a nodally exact solution. Through the introduction of some form of numerical dissipation, not only are we able to get smooth coarse grid solutions but, the resulting systems of linear equations are more amenable to iterative solutions methods. We now generalize the concept of upwind methods to the finite element framework.

Bibliography

[1] A. N. Brooks and T. J. R. Hughes. Streamline upwind/Petrov–Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier–Stokes equations. *Comp. Meth. Appl. Mech. Eng.*, 32:199–259, 1982.

[2] G. Fairweather, A. Gourlay, and A. Mitchell. Some high accuracy difference schemes with a splitting operator for equations of parabolic and elliptic type. *Numer. Math.*, 10:56–66, 1967.

[3] A. Hadjidimos. On some high accuracy difference schemes for solving elliptic equations. *Numeri. Math.*, 13:396–403, 1969.

[4] D. Peaceman and J. H. H. Rachford. The numerical solution of parabolic and elliptic differential equations. *J. Soc. Indust. Appl. Math.*, 3:28–41, 1955.